工程化专题之Maven

1. 认识Maven

2. 优势

1. 约定优于配置
2. 简单
3. 测试支持
4. 构建简单
5. CI
6. 插件丰富

3. 下载

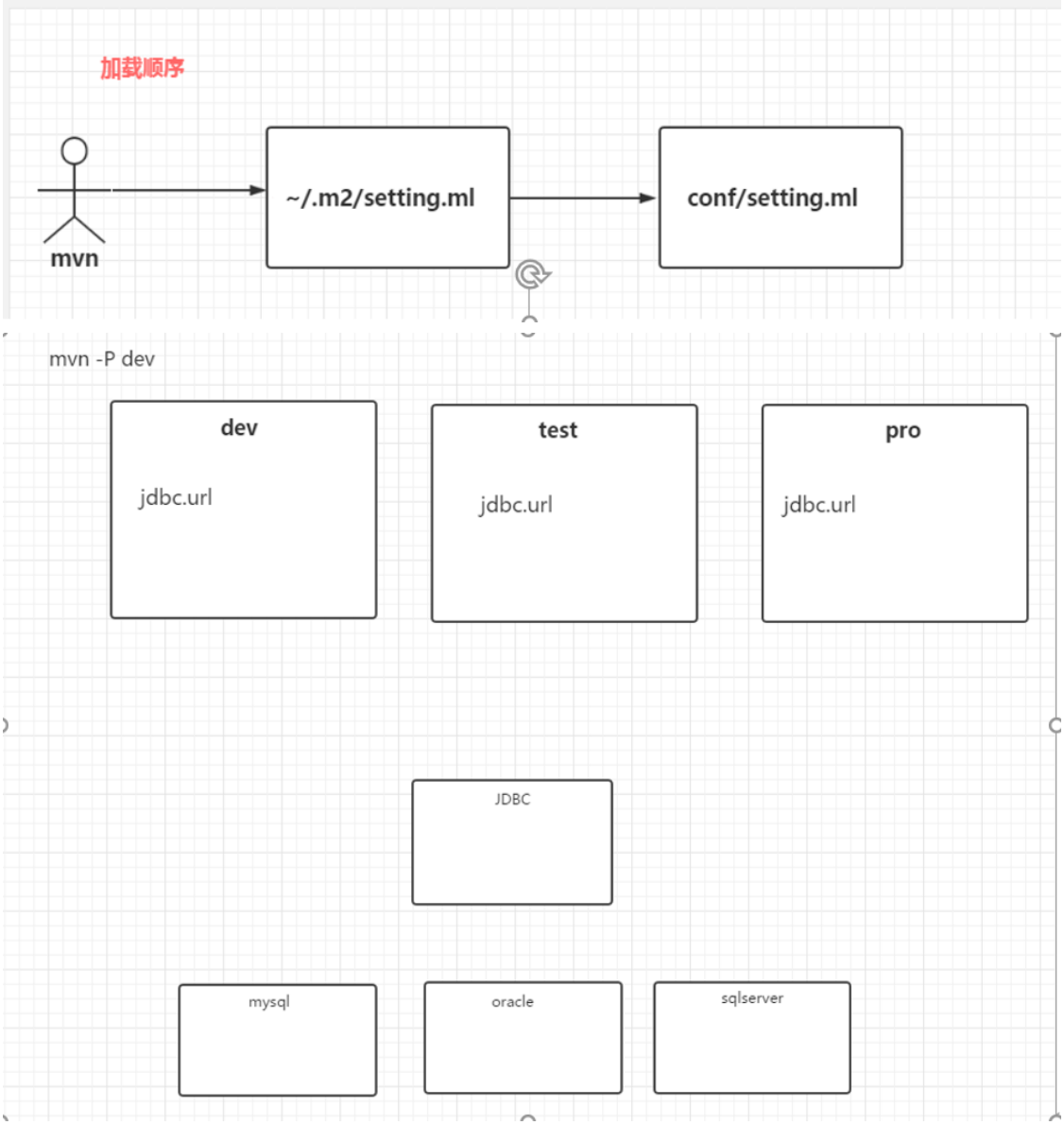
1. https://maven.apache.org/download.cgi
2. 安装
3. 超级pom文件：maven-model-builder-3.3.9.jar/org/apache/maven/model
4. 配置 MVM_HOME

   1）Windows 设置 path

   2）Linux 设置 .bash_profile

   3）设置 MAVEN_OPTS 能够直接重复使用

   4）配置 setting.xml：

- 使用 mvn 命令时，默认先使用 `C:\Users\Allen\.m2` 路径下找到setting.xml文件配置，如果找不到，就回去找maven安装目录下的 conf文件夹下的 setting.xml 进行配置
- ```
  <mirror>
      <id>alimaven</id>
      <name>aliyun maven</name>
      <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
      <mirrorOf>central</mirrorOf>
  </mirror>

  <mirror>
      <id>ui</id>
      <mirrorOf>central</mirrorOf>
      <name>Human Readable Name for this Mirror.</name>
      <url>http://uk.maven.org/maven2/</url>
  </mirror>

  <mirror>
      <id>osc</id>
      <mirrorOf>central</mirrorOf>
      <url>http://maven.oschina.net/content/groups/public/</url>
  </mirror>

  <mirror>
  ```
  ```

```
    <id>osc_thirdparty</id>
    <mirrorOf>thirdparty</mirrorOf>
    <url>http://maven.oschina.net/content/repositories/thirdparty/</url>
</mirror>
```

- 

maven 的 setting.xml 配置文件

- ```
  <?xml version="1.0" encoding="UTF-8"?>

  <!--
  Licensed to the Apache Software Foundation (ASF) under one
  or more contributor license agreements.  See the NOTICE file
  distributed with this work for additional information
  regarding copyright ownership.  The ASF licenses this file
  to you under the Apache License, Version 2.0 (the
  "License"); you may not use this file except in compliance
  with the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing,
  software distributed under the License is distributed on an
  "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
  KIND, either express or implied.  See the License for the
  ```
  ```

```
specific language governing permissions and limitations
under the License.
-->

<!--
 | This is the configuration file for Maven. It can be specified at two
levels:
 |
 |  1. User Level. This settings.xml file provides configuration for a
single user,
 |                 and is normally provided in
${user.home}/.m2/settings.xml.
 |
 |                 NOTE: This location can be overridden with the CLI
option:
 |
 |                 -s /path/to/user/settings.xml
 |
 |  2. Global Level. This settings.xml file provides configuration for all
Maven
 |                 users on a machine (assuming they're all using the same
Maven
 |                 installation). It's normally provided in
 |                 ${maven.conf}/settings.xml.
 |
 |                 NOTE: This location can be overridden with the CLI
option:
 |
 |                 -gs /path/to/global/settings.xml
 |
 | The sections in this sample file are intended to give you a running start
at
 | getting the most out of your Maven installation. Where appropriate, the
default
 | values (values used when the setting is not specified) are provided.
 |
 |-->
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository
  -->
  <localRepository>G:\blue_world\Documents\study-repository\repository-
springboot2.0.X</localRepository>


  <!-- 交互式模型 -->
  <!-- interactiveMode
   | This will determine whether maven prompts you when it needs input. If
set to false,
   | maven will use a sensible default value, perhaps based on some other
setting, for
   | the parameter in question.
   |
```

```
    | Default: true
  <interactiveMode>true</interactiveMode>
  -->

  <!-- offline
    | Determines whether maven should attempt to connect to the network when
executing a build.
    | This will have an effect on artifact downloads, artifact deployment,
and others.
    |
    | Default: false
  <offline>false</offline>
  -->

  <!-- maven的插件 -->
  <!-- pluginGroups
    | This is a list of additional group identifiers that will be searched
when resolving plugins by their prefix, i.e.
    | when invoking a command line like "mvn prefix:goal". Maven will
automatically add the group identifiers
    | "org.apache.maven.plugins" and "org.codehaus.mojo" if these are not
already contained in the list.
    |-->
  <pluginGroups>
    <!-- pluginGroup
      | Specifies a further group identifier to use for plugin lookup.
    <pluginGroup>com.your.plugins</pluginGroup>
    -->
  </pluginGroups>

  <!-- 配置代理 -->
  <!-- proxies
    | This is a list of proxies which can be used on this machine to connect
to the network.
    | Unless otherwise specified (by system property or command-line switch),
the first proxy
    | specification in this list marked as active will be used.
    |-->
  <proxies>
    <!-- proxy
      | Specification for one proxy, to be used in connecting to the network.
      |
    <proxy>
      <id>optional</id>
      <active>true</active>
      <protocol>http</protocol>
      <username>proxyuser</username>
      <password>proxypass</password>
      <host>proxy.host.net</host>
      <port>80</port>
      <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
    </proxy>
    -->
  </proxies>

  <!-- 搭建私服 -->
  <!-- servers
```

```
   | This is a list of authentication profiles, keyed by the server-id used
within the system.
   | Authentication profiles can be used whenever maven must make a
connection to a remote server.
   |-->
  <servers>
    <!-- server
     | Specifies the authentication information to use when connecting to a
particular server, identified by
     | a unique name within the system (referred to by the 'id' attribute
below).
     |
     | NOTE: You should either specify username/password OR
privateKey/passphrase, since these pairings are
     |       used together.
     |
    <server>
      <id>deploymentRepo</id>
      <username>repouser</username>
      <password>repopwd</password>
    </server>
    -->

    <!-- Another sample, using keys to authenticate.
    <server>
      <id>siteServer</id>
      <privateKey>/path/to/private/key</privateKey>
      <passphrase>optional; leave empty if not used.</passphrase>
    </server>
    -->
  </servers>

  <!-- mirrors
   | This is a list of mirrors to be used in downloading artifacts from
remote repositories.
   |
   | It works like this: a POM may declare a repository to use in resolving
certain artifacts.
   | However, this repository may have problems with heavy traffic at times,
so people have mirrored
   | it to several places.
   |
   | That repository definition will have a unique id, so we can create a
mirror reference for that
   | repository, to be used as an alternate download site. The mirror site
will be the preferred
   | server for that repository.
   |-->
  <mirrors>
    <!-- mirror
     | Specifies a repository mirror site to use instead of a given
repository. The repository that
     | this mirror serves has an ID that matches the mirrorOf element of
this mirror. IDs are used
     | for inheritance and direct lookup purposes, and must be unique across
the set of mirrors.
     |
    <mirror>
```

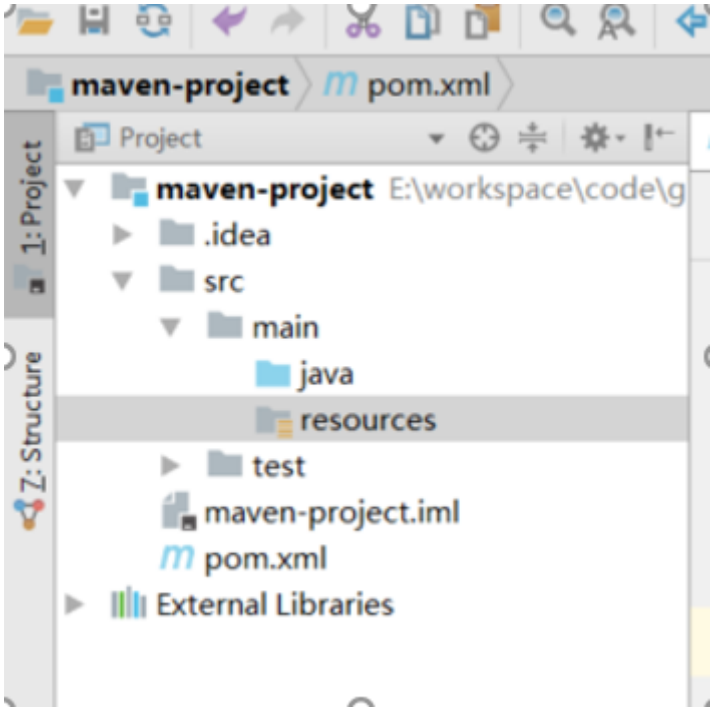
```
      <id>mirrorId</id>
      <mirrorOf>repositoryId</mirrorOf>
      <name>Human Readable Name for this Mirror.</name>
      <url>http://my.repository.com/repo/path</url>
    </mirror>
     -->

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

    <mirror>
        <id>alimaven</id>
        <mirrorOf>central</mirrorOf>
        <name>aliyun maven</name>

 <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
    </mirror>

  </mirrors>

  <!-- 用于配置环境的切换 -->
  <!-- profiles
   | This is a list of profiles which can be activated in a variety of ways,
and which can modify
   | the build process. Profiles provided in the settings.xml are intended
to provide local machine-
   | specific paths and repository locations which allow the build to work
in the local environment.
   |
   | For example, if you have an integration testing plugin - like cactus -
that needs to know where
   | your Tomcat instance is installed, you can provide a variable here such
that the variable is
   | dereferenced during the build process to configure the cactus plugin.
   |
   | As noted above, profiles can be activated in a variety of ways. One way
- the activeProfiles
   | section of this document (settings.xml) - will be discussed later.
Another way essentially
   | relies on the detection of a system property, either matching a
particular value for the property,
   | or merely testing its existence. Profiles can also be activated by JDK
version prefix, where a
   | value of '1.4' might activate a profile when the build is executed on a
JDK version of '1.4.2_07'.
   | Finally, the list of active profiles can be specified directly from the
command line.
   |
   | NOTE: For profiles defined in the settings.xml, you are restricted to
specifying only artifact
   |       repositories, plugin repositories, and free-form properties to be
used as configuration
   |       variables for plugins in the POM.
   |
```

```
   |-->
  <profiles>
    <!-- profile
     | Specifies a set of introductions to the build process, to be
activated using one or more of the
     | mechanisms described above. For inheritance purposes, and to activate
profiles via <activatedProfiles/>
     | or the command line, profiles have to have an ID that is unique.
     |
     | An encouraged best practice for profile identification is to use a
consistent naming convention
     | for profiles, such as 'env-dev', 'env-test', 'env-production', 'user-
jdcasey', 'user-brett', etc.
     | This will make it more intuitive to understand what the set of
introduced profiles is attempting
     | to accomplish, particularly when you only have a list of profile id's
for debug.
     |
     | This profile example uses the JDK version to trigger activation, and
provides a JDK-specific repo.
    <profile>
      <id>jdk-1.4</id>

      <activation>
        <jdk>1.4</jdk>
      </activation>

      <repositories>
        <repository>
          <id>jdk14</id>
          <name>Repository for JDK 1.4 builds</name>
          <url>http://www.myhost.com/maven/jdk14</url>
          <layout>default</layout>
          <snapshotPolicy>always</snapshotPolicy>
        </repository>
      </repositories>
    </profile>
    -->

    <!--
     | Here is another profile, activated by the system property 'target-
env' with a value of 'dev',
     | which provides a specific path to the Tomcat instance. To use this,
your plugin configuration
     | might hypothetically look like:
     |
     | ...
     | <plugin>
     |   <groupId>org.myco.myplugins</groupId>
     |   <artifactId>myplugin</artifactId>
     |
     |   <configuration>
     |     <tomcatLocation>${tomcatPath}</tomcatLocation>
     |   </configuration>
     | </plugin>
     | ...
     |
```

```
     | NOTE: If you just wanted to inject this configuration whenever
someone set 'target-env' to
     |       anything, you could just leave off the <value/> inside the
activation-property.
     |
    <profile>
      <id>env-dev</id>

      <activation>
        <property>
          <name>target-env</name>
          <value>dev</value>
        </property>
      </activation>

      <properties>
        <tomcatPath>/path/to/tomcat/instance</tomcatPath>
      </properties>
    </profile>
    -->
  </profiles>

  <!-- activeProfiles
   | List of profiles that are active for all builds.
   |
  <activeProfiles>
    <activeProfile>alwaysActiveProfile</activeProfile>
    <activeProfile>anotherAlwaysActiveProfile</activeProfile>
  </activeProfiles>
  -->
</settings>
```

4. 新建一个Maven项目

4.1项目结构



4.2 pom.xml

1. groupId ： com.gupaoedu
2. artfactId ： 功能命名
3. version ： 版本号
4. packaging： 打包方式，默认是jar
5. **dependencyManagement：**

    a. 只能出现在父pom (约定，不是必须的)

    b. 统一版本号

    c. **声明 (子POM里用到再引)，在父pom文件中使用dependencyManagement 管理的依赖，其子项目不会直接拥有这个依赖；如果想在子项目中使用该依赖，需要在子pom中重新引用**

6. dependency：

    a. Type 默认jar

    b. scope：

    1) compile ： 默认的，编译时使用，会将管理的此包打到项目的包中，例如spring的spring-core

    2) test ： 测试时使用该包

    3) provided：编译时使用，但是打包项目时，不会讲此包导入到项目包，例如tomcat的servlet

    4) runtime： 运行时使用，编译无效，例如JDBC驱动实现

    5) system ： 本地一些jar包（或自己私服的jar包），不需要放入到中央仓库中管理，例如短信jar

    6) scope是有 **依赖传递** 性的：

- ◦ 第一列表示直接依赖的scope，第一行表示间接依赖的scope

	compile	test	provided	runtime
compile	compile	-	-	runtime
test	test	-	-	test
provided	provided	-	provided	provided
runtime	runtime	-	-	runtime

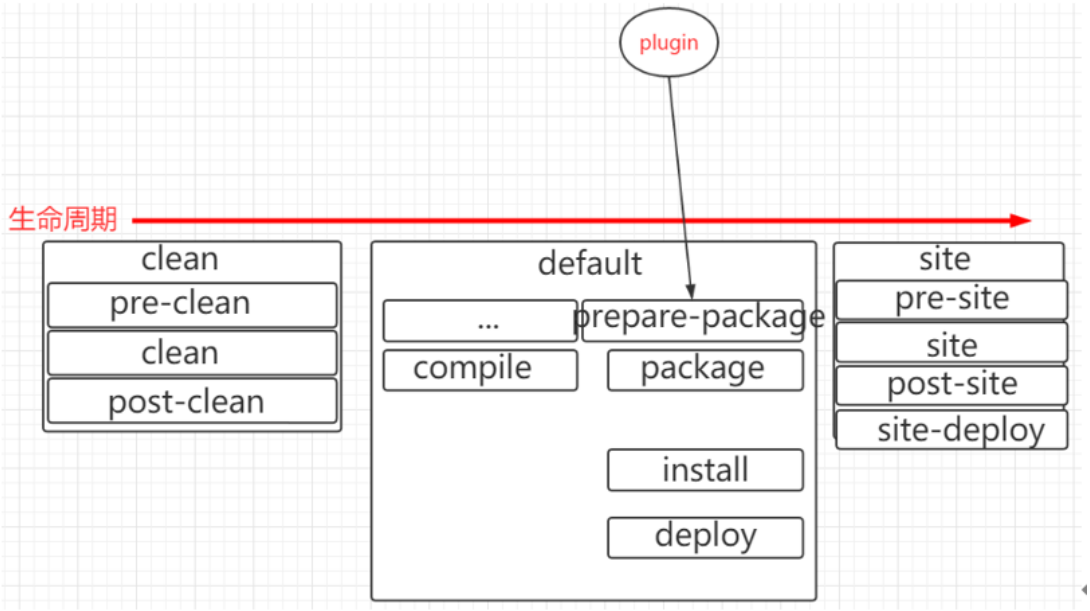
7) 依赖仲裁

i. 最短路径原则：

ii. 加载先后原则：在路径上是一样时，根据pom文件中书写顺序进行加载先后排序

8) exclusions：排除包

7. 生命周期 lifecycle/phase/goal



- A Build Lifecycle is Made Up of Phases
- A Build Phase is Made Up of Plugin Goals

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

- 一个生命周期可以有很多phase(阶段)，每个阶段又有许多的插件的运行
- 流程化、责任链可以参考maven，将生命周期抽象出来

5. 版本管理

1. 1.0-SNAPSHOT （maven默认的版本号，不稳定的版本）

- repository 删除
- **mvn clean package -U** (不管本地仓库是否有这个jar包，都会强制拉取一次)
- SNAPSHOP版本的jar包上传私服可以重复上传，RELEASE版本不可以

2. 主版本号.次版本号.增量版本号-<里程碑版本>

- 1.0.0-RELAESE

6. 常用命令

1. compile： 编译
2. clean ： 删除 target/
3. test ： test case，包括 junit 和 testNG
4. package：打包
5. install ： 把项目 install 到 local repo
6. deploy ： 把本地 jar 发布到 remote（私服）

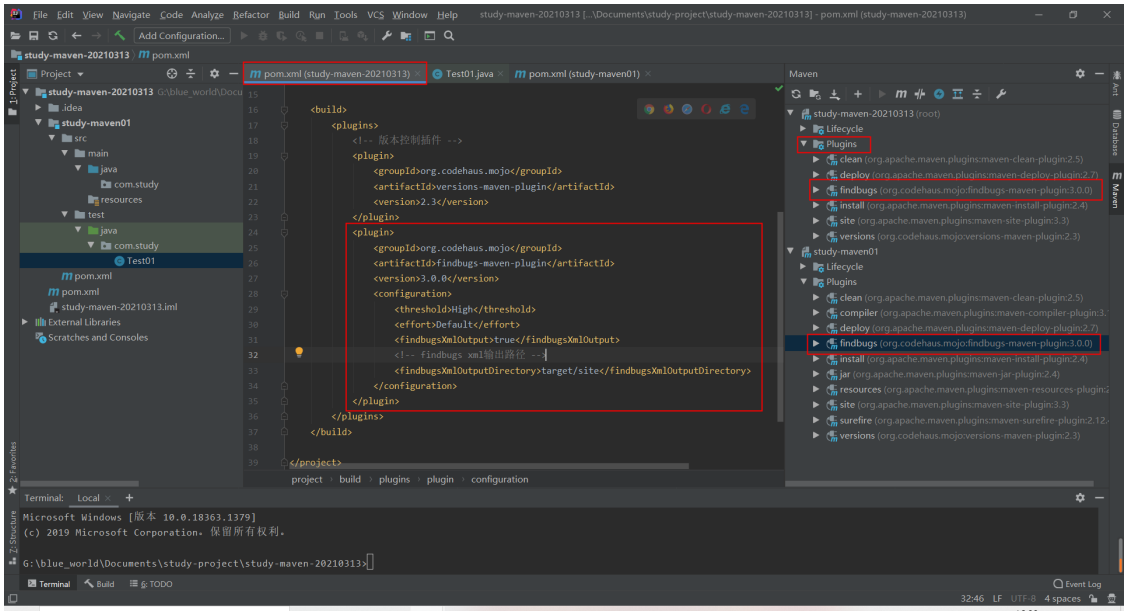
7. 插件

1. 常用插件下载地址

- maven：https://maven.apache.org/plugins/
- majohaus：http://www.mojohaus.org/plugins.html

2. 常用的插件：

- findbugs：静态代码检查



```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>findbugs-maven-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <threshold>High</threshold>
        <effort>Default</effort>
        <findbugsXmlOutput>true</findbugsXmlOutput>
        <!-- findbugs xml输出路径 -->
        <findbugsXmlOutputDirectory>target/site</findbugsXmlOutputDirectory>
    </configuration>
</plugin>
```

- versions：统一升级版本号

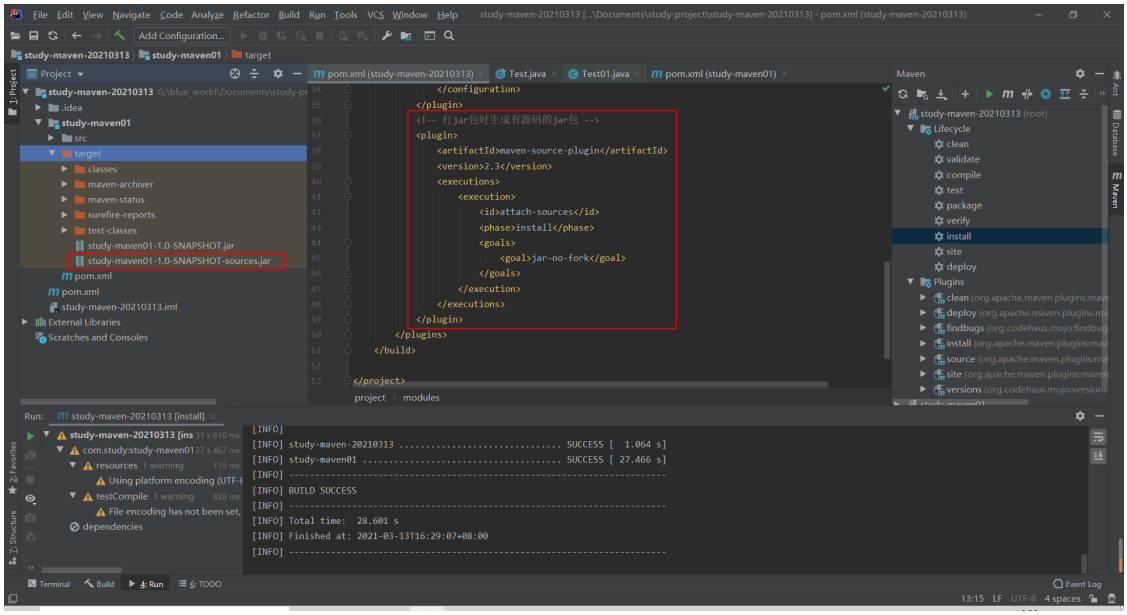
  命令：`mvn versions:set -DnewVersion=1.1`

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>versions-maven-plugin</artifactId>
    <version>2.3</version>
</plugin>
```

- source 打包源代码

  命令：`source 打包源代码`

```
<!-- 打jar包时生成有源码的jar包 -->
<plugin>
    <artifactId>maven-source-plugin</artifactId>
    <version>2.3</version>
    <executions>
        <execution>
        <id>attach-sources</id>
        <phase>install</phase>
        <goals>
            <goal>jar-no-fork</goal>
        </goals>
        </execution>
    </executions>
</plugin>
```



- assembly 打包zip、war
- tomcat7

8. 自定义插件

- https://maven.apache.org/guides/plugin/guide-java-plugin-development.html

1. pom中配置：`<packaging>maven-plugin</packaging>`

```
<?xml version="1.0" encoding="UTF-8"?>
<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/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.study</groupId>
    <artifactId>study-maven-plugin</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>maven-plugin</packaging>

    <dependencies>
        <dependency>
            <groupId>org.apache.maven</groupId>
            <artifactId>maven-plugin-api</artifactId>
            <version>3.5.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.plugin-tools</groupId>
            <artifactId>maven-plugin-annotations</artifactId>
            <version>3.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

</project>
```

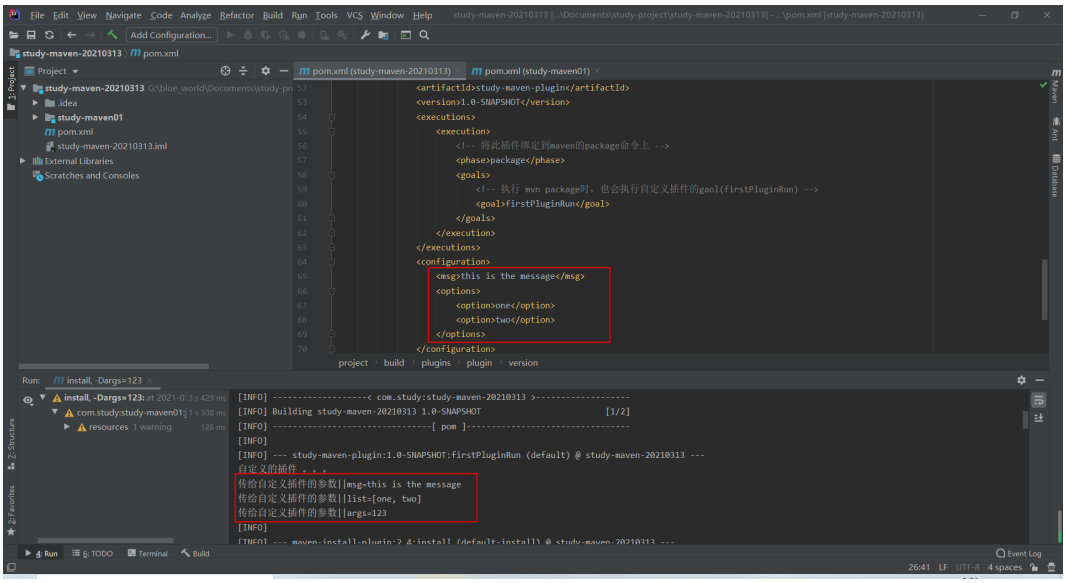
2. 自定义的插件类需要 **extends AbstractMojo**

```
package com.study;

import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;
import org.apache.maven.plugin.MojoFailureException;
import org.apache.maven.plugins.annotations.LifecyclePhase;
import org.apache.maven.plugins.annotations.Mojo;
import org.apache.maven.plugins.annotations.Parameter;

import java.util.List;

/**
 * 自定义插件需要继承 AbstractMojo 类
 * 使用 name的值 来运行这个插件
 */
@Mojo(name = "firstPluginRun", defaultPhase = LifecyclePhase.PACKAGE)
public class GroupMojo extends AbstractMojo {

    @Parameter
    private String msg;

    @Parameter
    private List<String> options;

    @Parameter(property = "args")
    private String args;

    public void execute() throws MojoExecutionException,
MojoFailureException {
        System.out.println("自定义的插件 。。。 ");
        System.out.println("传给自定义插件的参数||msg=" + msg);
        System.out.println("传给自定义插件的参数||list=" + options);
        System.out.println("传给自定义插件的参数||args=" + args);
    }

}
```

3. 使用插件的项目中进行配置`<plugin>`

```
<!-- 使用自定义的插件 -->
<plugin>
    <groupId>com.study</groupId>
    <artifactId>study-maven-plugin</artifactId>
    <version>1.0-SNAPSHOT</version>
    <executions>
        <execution>
            <!-- 将此插件绑定到maven的package命令上 -->
            <phase>package</phase>
            <goals>
                <!-- 执行 mvn package时，也会执行自定义插件的
gaol(firstPluginRun) -->
                <goal>firstPluginRun</goal>
            </goals>
        </execution>
```

```
    </executions>
    <configuration>
        <msg>this is the message</msg>
        <options>
            <option>one</option>
            <option>two</option>
        </options>
    </configuration>
</plugin>
```

4. 执行 mvn package 查看控制台中自定义插件是否执行
5. 参数传递
    - 执行maven命令：`mvn package -Dargs=123`
    - 

9. Profile

1. 使用场景 dev/test/pro
2. setting.xml 家和公司两套

settings.xml

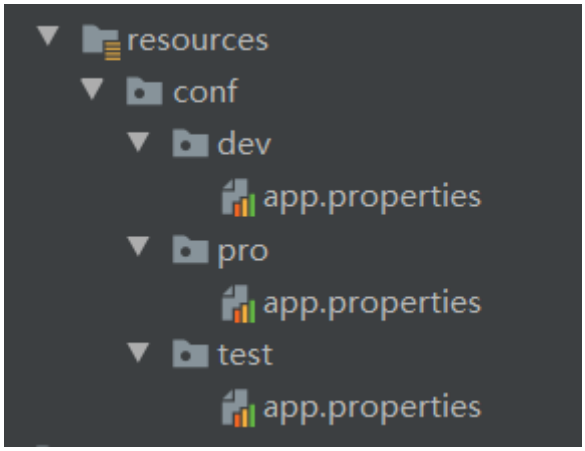
```
265       </properties>
266     </profile>
267     -->
268
269     <!-- 公司私服仓库地址配置-->
270     <profile>
271       <id>companyEnvironment</id>
272       <repositories>
273         <repository>
274             <id>repo-mirro</id>
275             <url>http://uk.maven.org/maven/</ur/>
276         </repository>
277       </repositories>
278       <pluginRepositories>
279         <pluginRepositorie>
280             <id>plugin-repo-mirror</id>
281             <url>http://uk.maven.org.maven</url>
282         </pluginRepositorie>
283       </pluginRepositories>
284     </profile>
285     <!-- 家庭办公时仓库地址配置-->
286     <profile>
287       <id>homeEnvironment</id>
288       <repositories>
289         <repository>
290             <id>repo-mirro</id>
291             <url>http://x'x'x/maven/</ur/>
292         </repository>
293       </repositories>
294       <pluginRepositories>
295         <pluginRepositorie>
296             <id>plugin-repo-mirror</id>
297             <url>http://xxx.maven</url>
298         </pluginRepositorie>
299       </pluginRepositories>
300     </profile>
301
302   </profiles>
303
304   <activeProfiles>
305     <activeProfile>homeEnvironment</activeProfile>
306   </activeProfiles>
307
308   <!-- activeProfiles
```

3. 使用方法：
   - 配置Pom文件

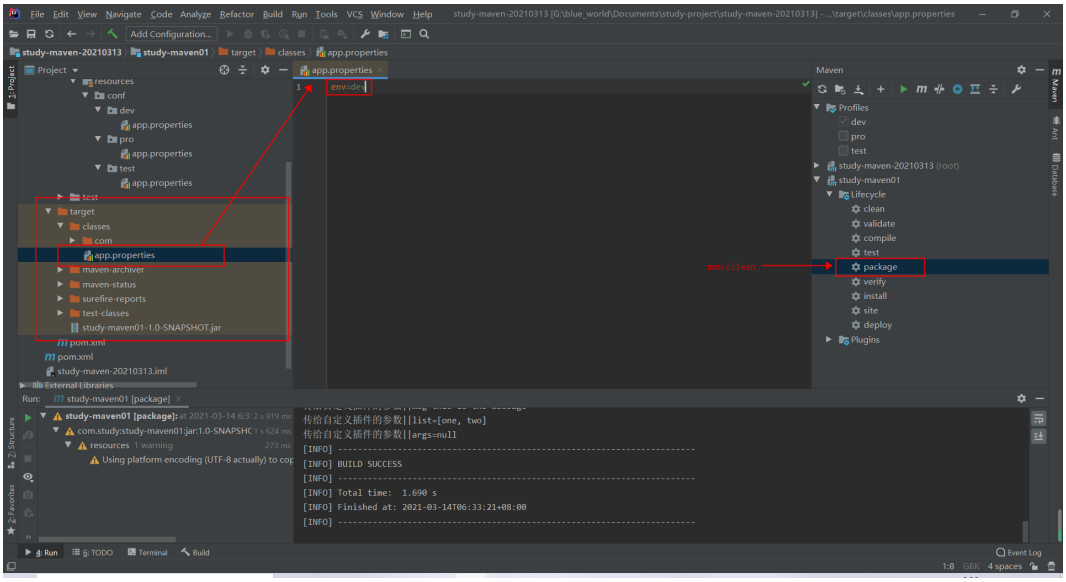
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>study-maven-20210313</artifactId>
        <groupId>com.study</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <packaging>jar</packaging>

    <artifactId>study-maven01</artifactId>

    <!-- 设置profile属性 -->
```

```
    <profiles>
        <!-- 开发环境配置文件所在位置 -->
        <profile>
            <id>dev</id>
            <properties>
                <profiles.active>dev</profiles.active>
            </properties>
            <activation>
                <!-- 默认激活环境为dev -->
                <activeByDefault>true</activeByDefault>
            </activation>
        </profile>
        <!-- 测试环境配置文件所在位置 -->
        <profile>
            <id>test</id>
            <properties>
                <profiles.active>test</profiles.active>
            </properties>
        </profile>
        <!-- 生产环境配置文件所在位置 -->
        <profile>
            <id>pro</id>
            <properties>
                <profiles.active>pro</profiles.active>
            </properties>
        </profile>
    </profiles>

    <!-- 在build中使用profile属性-->
    <build>
        <resources>
            <resource>
                 <!--${basedir}就是工程的主目录-->
                <directory>${basedir}/src/main/resources</directory>
                <excludes>
                    <!-- 将主目录下所有conf文件夹下的文件都去除掉-->
                    <exclude>conf/**</exclude>
                </excludes>
            </resource>
            <resource>
                <!-- 保留下面路径的文件 -->
                <directory>src/main/resources/conf/${profiles.active}
</directory>
            </resource>
        </resources>
    </build>

</project>
```

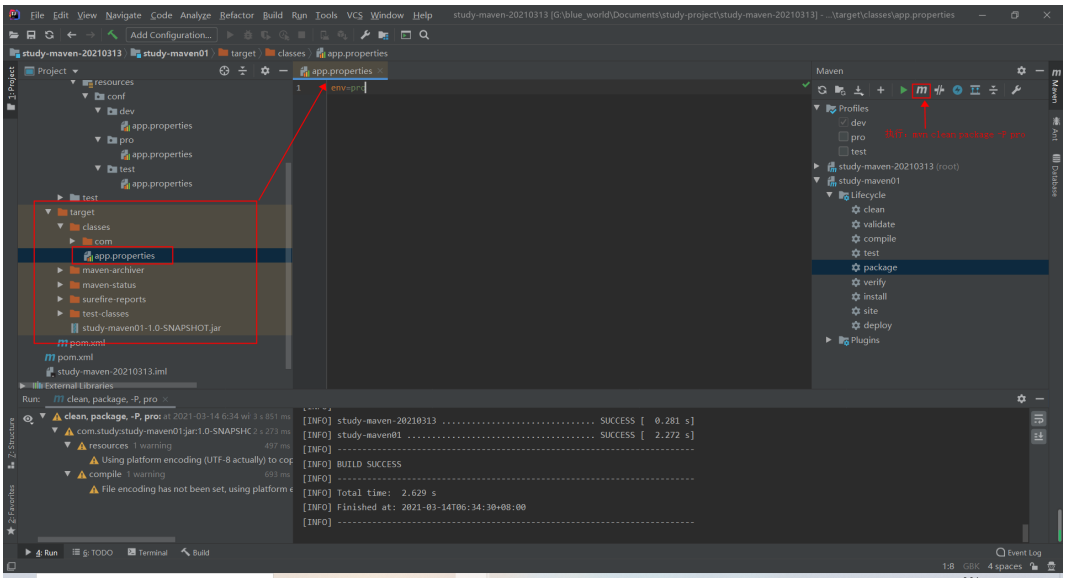
- 设置配置文件文件



- 执行命令：
  - **mvn clean package**



  - **mvn clean package -P pro**



10. 仓库（配置私服）

1. 下载 https://www.sonatype.com/nexus-repository-oss
2. 安装 解压
3. 使用

- 本地启动nexus（或者服务器上启动nexus）

    - cmd中执行：`nexus.exe /run` 指令
    - 第一次需要，安装服务：
        - 以管理员权限进入cmd，执行 `nexus.exe /install`
        - 接着启动服务 `net start nexus`
        - 然后停止服务 `net stop nexus`
    - 以后启动服务，只要执行 `nexus.exe /run` 指令 即可
- 登录 http://192.168.1.6:8081/nexus 查看 nexus服务

4. 发布

- 在pom.xml中配置，上传资源到nexus

```
<distributionManagement>
    <repository>
        <id>nexus-release</id>
        <name>Nexus Release Repository</name>
        <!-- 登录nexus,从上面获取url地址 -->
        <url>http://localhost:8081/repository/maven-releases/</url>
    </repository>
    <snapshotRepository>
        <id>nexus-snapshots</id>
        <name>Nexus Snapshot Repository</name>
        <url>http://localhost:8081/repository/maven-snapshots/</url>
    </snapshotRepository>
</distributionManagement>
```

- 在maven的settings.xml配置文件中配置上传权限

```
<servers>
    <server>
        <id>nexus-release</id>
        <username>admin</username>
        <password>sw19941021</password>
    </server>
    <server>
        <id>nexus-snapshots</id>
        <username>admin</username>
        <password>sw19941021</password>
    </server>
</servers>
```

5. 从私服上拉取 jar

- 配置 mirror

```
 <mirror>
     <id>nexus</id>
     <mirrorOf>*</mirrorOf>
     <url>http://localhost:8081/repository/maven-public/</url>
 </mirror>
```

- 配置 Profile

```
<profile>
```

```
    <id>nexus</id>
    <repositories>
        <repository>
            <id>central</id>
            <url>https://repo1.maven.org/maven2</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>central</id>
            <url>https://repo1.maven.org/maven2</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
</profile>
```

- setting.xml 中生效 profile 配置

```
<activeProfiles>
    <activeProfile>nexus</activeProfile>
</activeProfiles>
```

11. archetype 模版化

1. 生成一个archetype

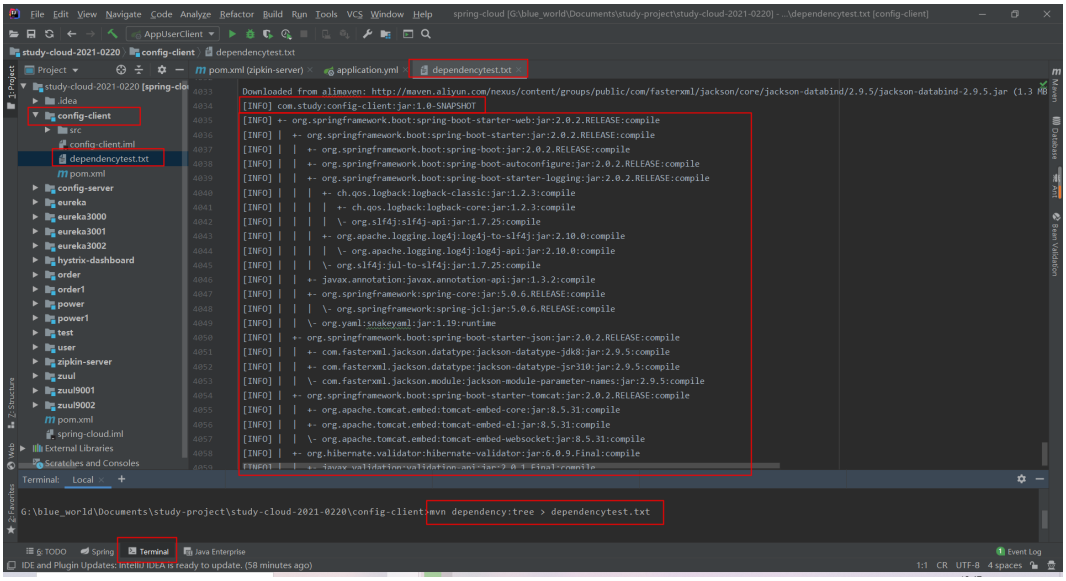
- mvn archetype:create-from-project
- cd /target/generated-sources/archetype
- mvn install

2. 从archetype创建项目

- mvn archetype:generate -DarchetypeCatalog=local

知识点：

1. maven 查看项目依赖关系命令：
    - `mvn dependency:tree > 文件名.txt`



2. 查看系统变量：
    - `mvn help:system`

课后作业：

1. 实战插件：
    - 给自己的 js\css 文件做处理防止缓存
    - 统计项目里面有多少个 java 文件
2. 实战archtype
3. 配置自己的私服