

day27_Maven

一 内容回顾 (列举前一天重点难点内容)

1.1 教学重点:

- 1 1. 掌握事务的简单使用
- 2 2. 掌握事务的特性
- 3 3. 掌握事务的隔离级别
- 4 4. 掌握常用的三方连接池
- 5 5. 掌握DAO设计模式
- 6 6. 熟练使用DButils实现增删改
- 7 7. 熟练使用DButils实现模型封装
- 8 8. 掌握xml和json的简单编写

1.2 教学难点:

- 1 1. 数据源的实现原理
- 2 2. ResultSetHandler接口的实现原理
- 3 3. xml约束的实现

二 教学目标

- 1 1. 熟悉maven的原理
- 2 2. 熟悉为什么使用maven
- 3 3. 熟练掌握idea与maven的联合使用
- 4 4. 熟练掌握maven配置pom.xml
- 5 5. 了解eclipse与maven的联合使用
- 6 6. 了解项目构建

三 教学导读

3.1. Maven工具的简介

3.1.1. 开发过程中遇到的问题

- 1 1、都是同样的代码，为什么在我的机器上可以编译执行，而在他的机器上就不行？
- 2 2、为什么在我的机器上可以正常打包，而配置管理员却打不出来？
- 3 3、项目组加入了新的人员，我要给他说明编译环境如何设置，但是让我挠头的是，有些细节我也记不清楚了。
- 4 4、我的项目依赖一些jar包，我应该把他们放哪里？放源码库里？
- 5 5、这是我开发的第二个项目，还是需要上面的那些jar包，再把它们复制到我当前项目的svn库里吧
- 6 6、现在是第三次，再复制一次吧。 ----- 这样真的好吗？
- 7 7、我写了一个数据库相关的通用类，并且推荐给了其他项目组，现在已经有五个项目组在使用它了，今天我发现了一个bug，并修正了它，我会把jar包通过邮件发给其他项目组 -----这不是一个好的分发机制，太多的环节可能导致出现bug
- 8 8、项目进入测试阶段，每天都要向测试服务器部署一版。每次都手动部署，太麻烦了。

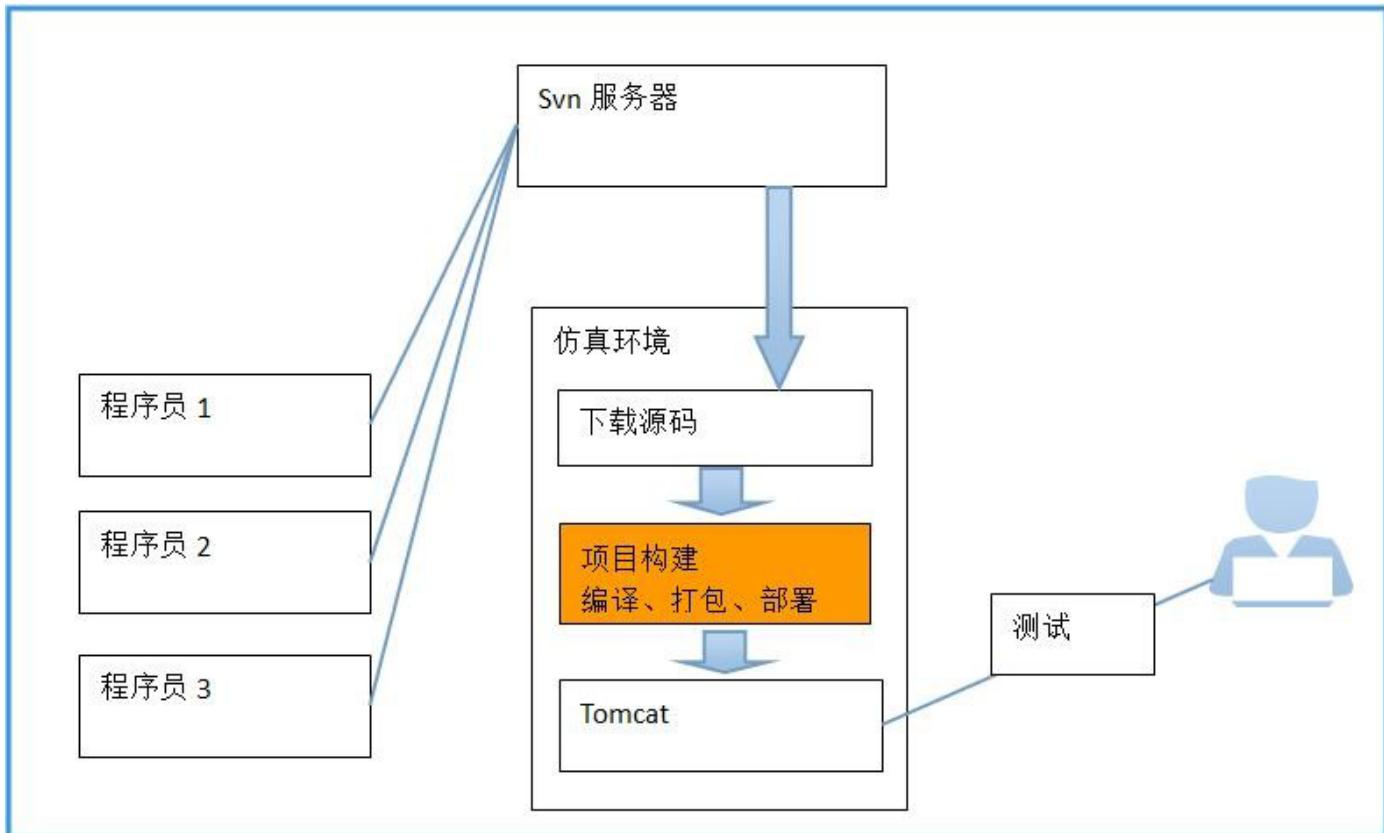
3.1.2. 什么是Maven (会)

Maven是基于项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。Maven是跨平台的项目管理工具。主要服务于基于Java平台的项目构建，依赖管理和项目信息管理。

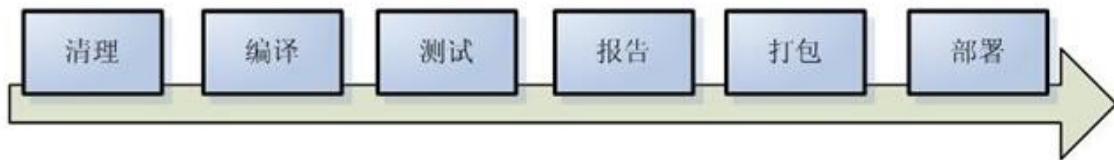
Maven主要有两个功能：

- 1 - 项目构建
- 2 - 依赖管理

3.1.3. 什么是构建(了解)



构建过程:



3.1.4 项目构建的方式

- Eclipse

1 | 手工操作较多，项目的构建过程都是独立的，很难一步完成。比如：编译、测试、部署等。开发时每个人的IDE配置都不同，很容易出现本地代码换个地方编译就出错

- Ant

1 | Ant只是一个项目构建工具，它没有集成依赖管理。Ant在进行项目构建时，它没有对项目目录结构进行约定，需要手动指定源文件、类文件等目录地址。同时它执行task时，需要显示指定依赖的task，这样会造成大量的代码重复。

- Maven

1 | - Maven不仅是一个项目构建工具，更是一个项目管理工具。它在项目构建工程中，比ant更全面，更灵活。
2 | - Maven在进行项目构建时，它对项目目录结构拥有约定，知道你的源代码在哪里，类文件应该放到哪里去。
3 | - 它拥有生命周期的概念，maven的生命周期是有顺序的，在执行后面的生命周期的任务时，不需要显示的配置前面任务的生命周期。例如执行 mvn install 就可以自动执行编译，测试，打包等构建过程

四 教学内容

4.1. Maven工具使用

4.1.1. Maven工具的安装 (会)

4.1.1.1. 环境准备

1 | PC端：win10/win7操作系统
2 | Jdk：jdk-8u172-windows-x64.exe
3 | maven：apache-maven-3.5.3-bin.zip

Maven的历史版本下载地址: <https://archive.apache.org/dist/maven/maven-3/>

4.1.1.2 maven解压

将Maven的软件包解压到某一个盘符下，如D盘，E盘的某一个你喜欢的路径下;如果你C盘的空间足够大，也可以选择C盘。注意：你的路径中不要使用中文，也尽量不要带有空格及其他特殊符号

例:如图所示:

此电脑 > 新加卷 (D:) > apache-maven-3.5.3 >			
名称	修改日期	类型	大小
bin	2019/12/28 下午2:15	文件夹	
boot	2019/12/28 下午2:15	文件夹	
conf	2019/12/28 下午2:15	文件夹	
lib	2019/12/28 下午2:15	文件夹	
LICENSE	2018/2/25 上午3:51	文件	21 KB
NOTICE	2018/2/25 上午3:51	文件	1 KB
README.txt	2018/2/25 上午3:46	文本文档	3 KB

4.1.1.3 Maven本地仓库配置

3.1 创建仓库目录

选择一个你喜欢的位置创建一个文件夹repository，当作Maven的本地仓库。起名尽量见名知意,这里的仓库是maven1/repository

例:如图所示

此电脑 > 新加卷 (D:) > maven1 >			
名称	修改日期	类型	大小
repository	2019/9/9 上午11:05	文件夹	

3.2 修改settings.xml文件

第一步:打开Maven的conf目录下的settings.xml文件

此电脑 > 新加卷 (D:) > apache-maven-3.5.3 > conf >			
名称	修改日期	类型	大小
logging	2019/12/28 下午2:15	文件夹	
settings.xml	2019/12/28 下午2:22	XML 文档	11 KB
settings.xml.bak	2018/2/25 上午3:46	BAK 文件	10 KB
toolchains.xml	2018/2/25 上午3:46	XML 文档	4 KB

第二步:找到localRepository标记，大概在文件的50行左右，然后配置一下本地仓库路径，指向你创建的仓库目录

例如如图所示

```
<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 ← 找到此标记
    | The path to the local repository maven will use to store artifacts.
    |
    | Default: ${user.home}/.m2/repository
    <localRepository>/path/to/local/repo</localRepository>
    <localRepository>D:/maven1/repository</localRepository>
    </!-- interactiveMode -->
```

第三步:再配置一下远程仓库的地址。下面给大家的是阿里云的远程仓库地址。

```
1 <mirror>
2     <id>alimaven</id>
3     <name>aliyun maven</name>
4
5     <url>http://maven.aliyun.com/nexus/content/groups/public
6     </url>
7         <mirrorOf>central</mirrorOf>
8     </mirror>
```

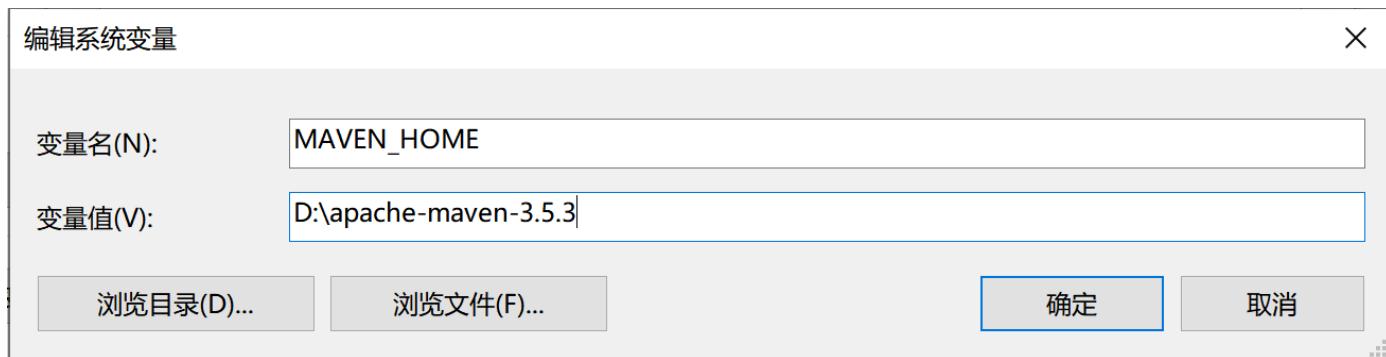
只需要将上述提供的阿里云的地址复制到mirrors标记内部即可，mirrors的结束标记大约在165行左右

```
156     <url>http://my.repository.com/repo/path</url>
157     </mirror>-->
158
159     <mirror>
160         <id>alimaven</id>
161         <name>aliyun maven</name>
162         <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
163         <mirrorOf>central</mirrorOf>
164     </mirror>
165 </mirrors> ← 找到mirrors的结束标记，将内容粘贴到里面
```

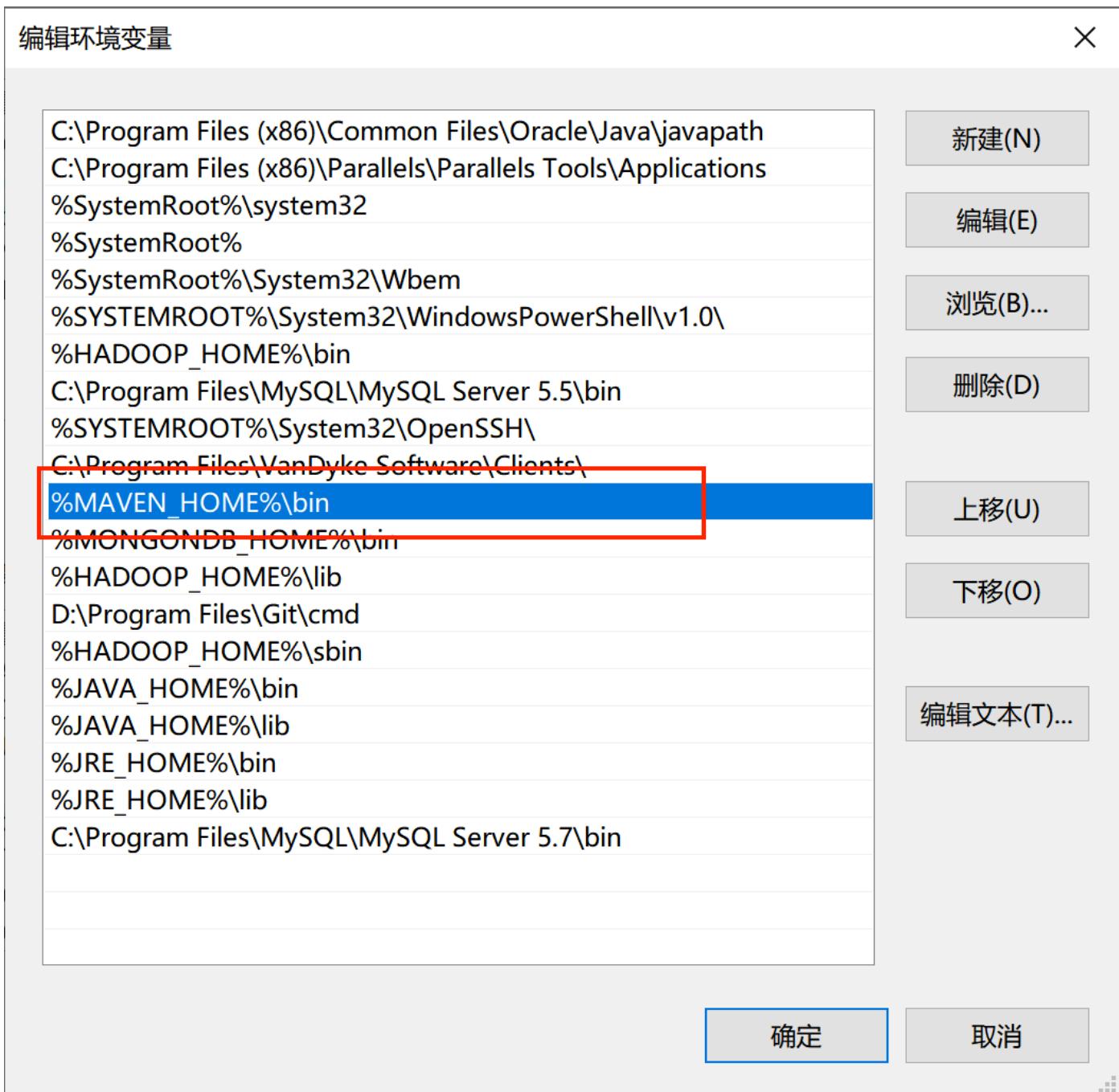
4.1.1.4. Maven环境变量配置

Maven的环境变量配置与jdk的环境变量配置非常类似。

4.1. 添加MAVEN_HOME变量



4.2. 配置path变量



4.3 验证环境变量

```
C:\WINDOWS\system32>mvn -version
Apache Maven 3.5.3 (3383c37e1f9e9b3bc3df5050c29c8aff9f295297; 2018-02-25T03:49:05+08:00)
Maven home: D:\apache-maven-3.5.3\bin\..
Java version: 1.8.0_221, vendor: Oracle Corporation
Java home: D:\ProgramFiles\Java\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
C:\Windows\System32>
```

出现类似上述的内容，表示配置成功

4.1.2. IntelliJ IDEA与Maven工具的整合(会)

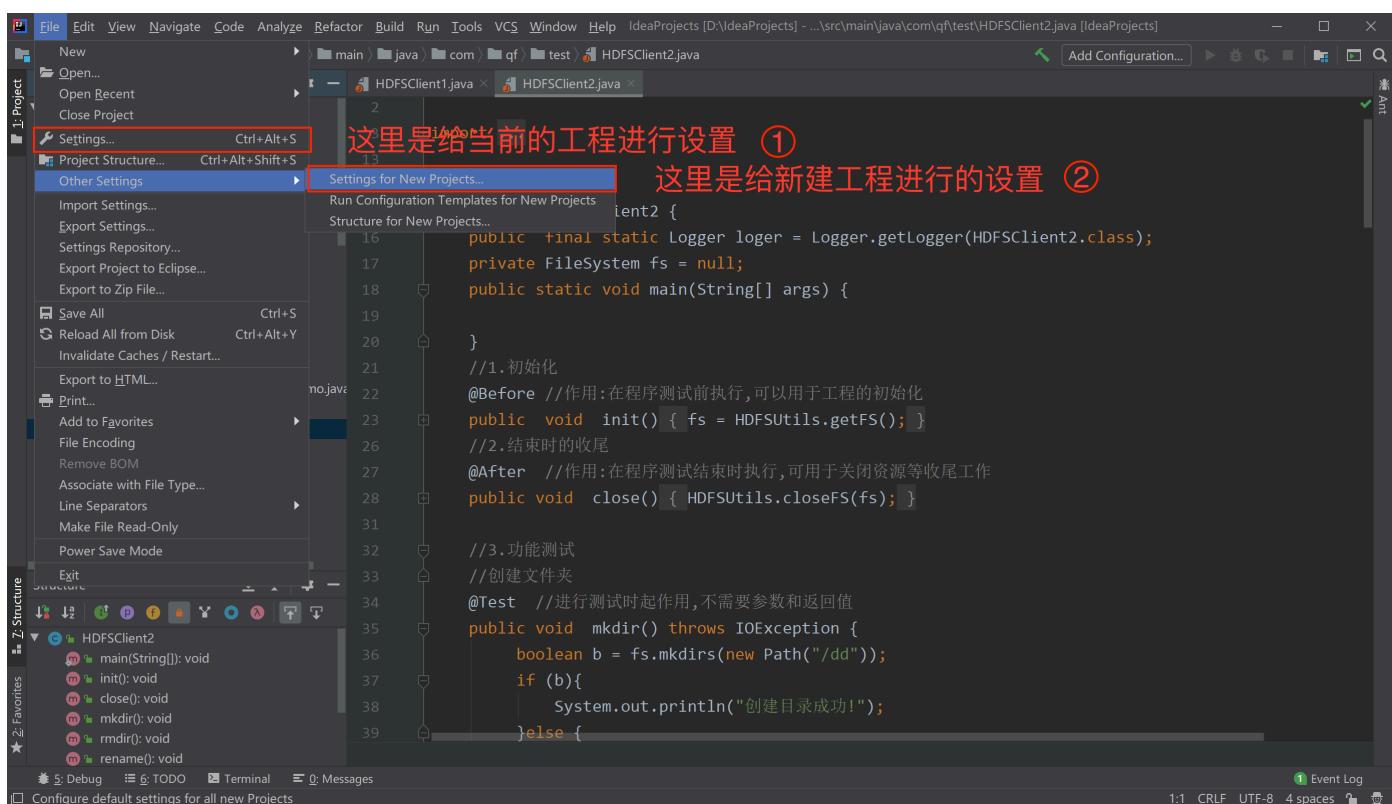
4.1.2.1 Idea配置Maven

1.1 打开idea,选择File,在下拉列表中会看到Settings和Other Settings.如下图

Settings和Other Settings.的区别:

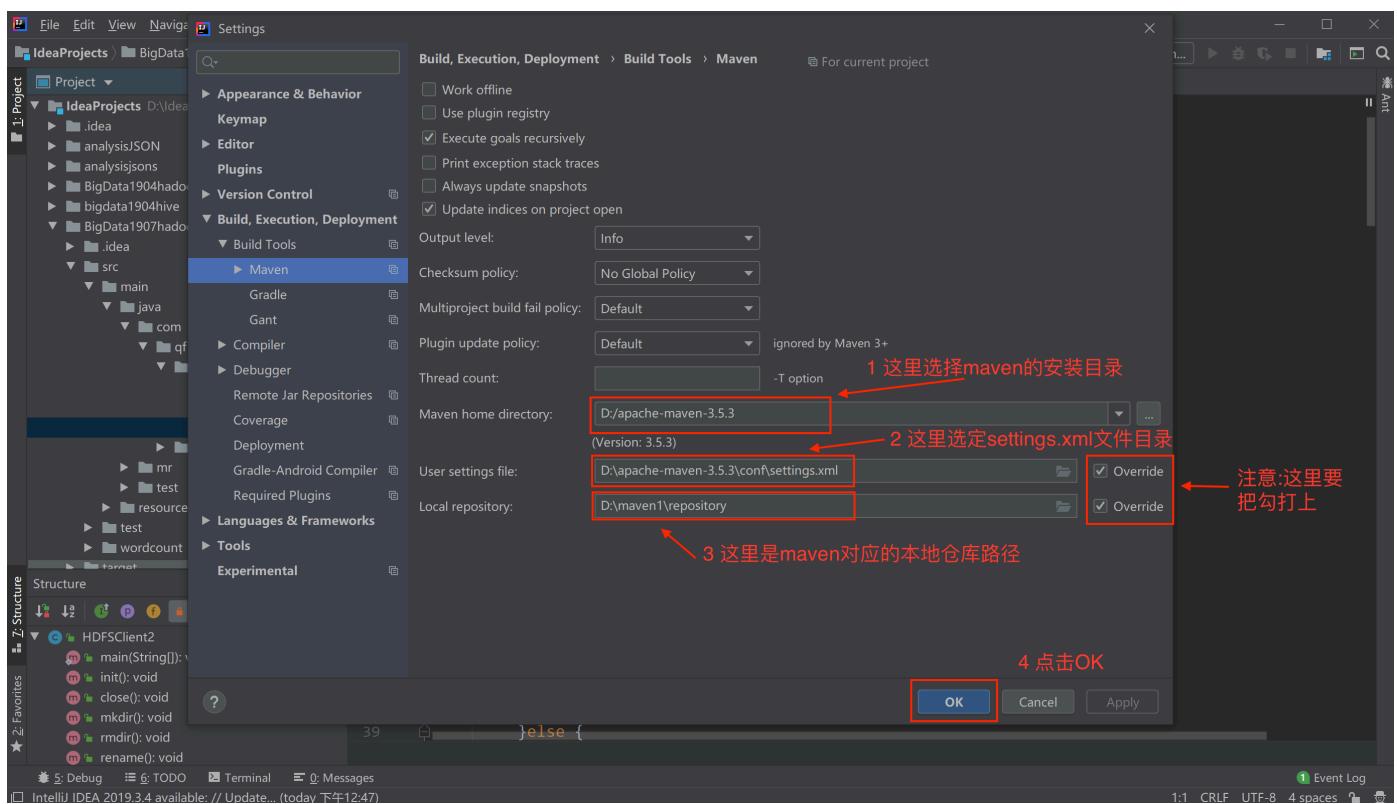
Settings是给当前的工程进行设置

Other Settings是给新建工程进行的设置



1.2 点击Settings或Other Settings都可以进入下图的配置界面，在这里指定maven的根目录,使用的settings.xml配置文件,并查看本地仓库的路径是否正确。

- 1 信息说明：
- 2
- 3 手动打开Maven配置界面路径：File--Settings--Build, Execution, Deployment—Build Tools—Maven
- 4
- 5 Maven home directory：当前Maven的安装路径
- 6
- 7 User Settings File：当前Maven内部的settings.xml文件，要将Override打钩
- 8
- 9 Local repository：当前Maven对应的本地仓库地址，要将Override打钩

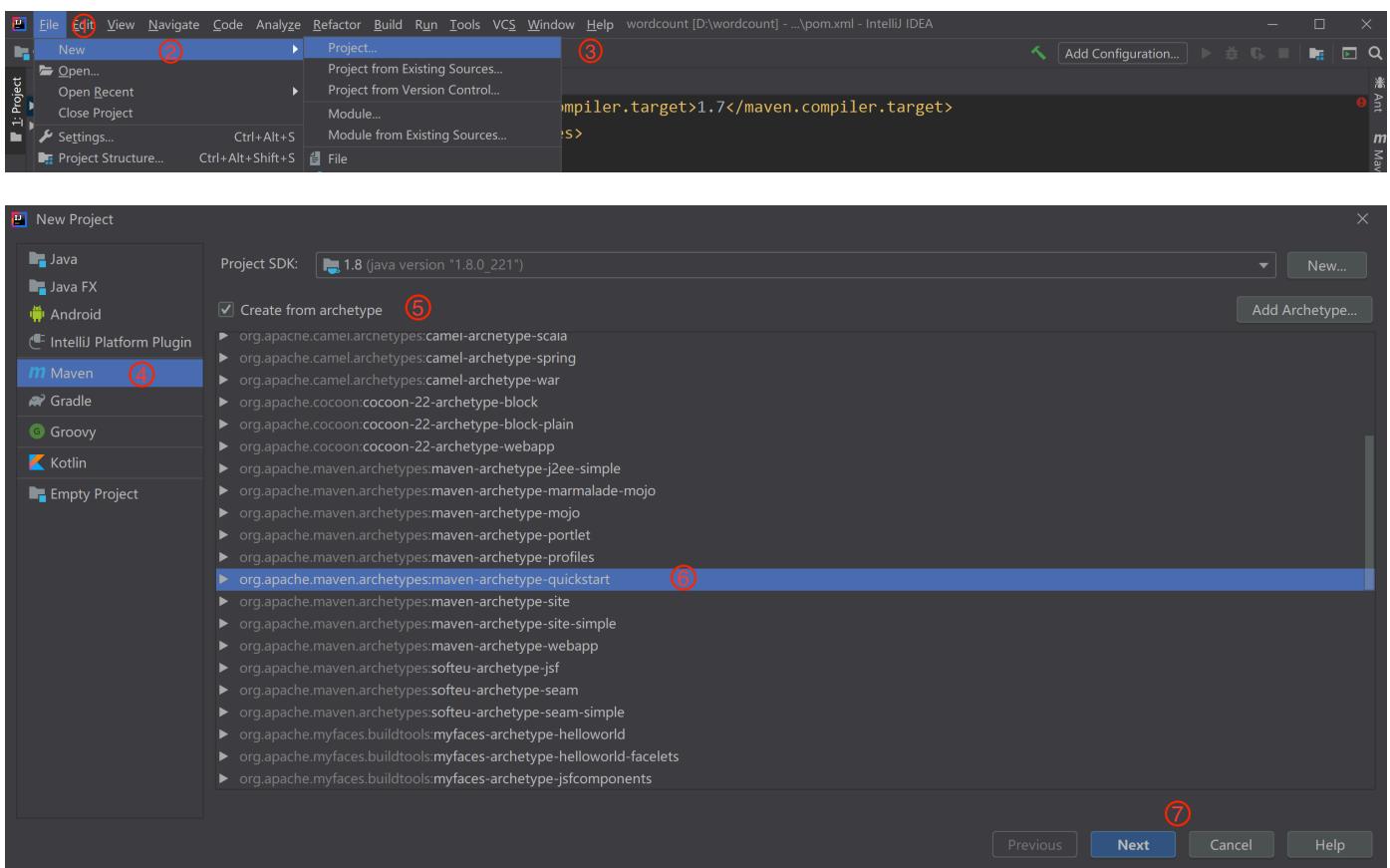


4.1.2.2 创建Maven项目,下载通用jar包

选择 File --> New --> Project --> Maven, 勾上create from archetype, 选择quickstart, 然后Next。

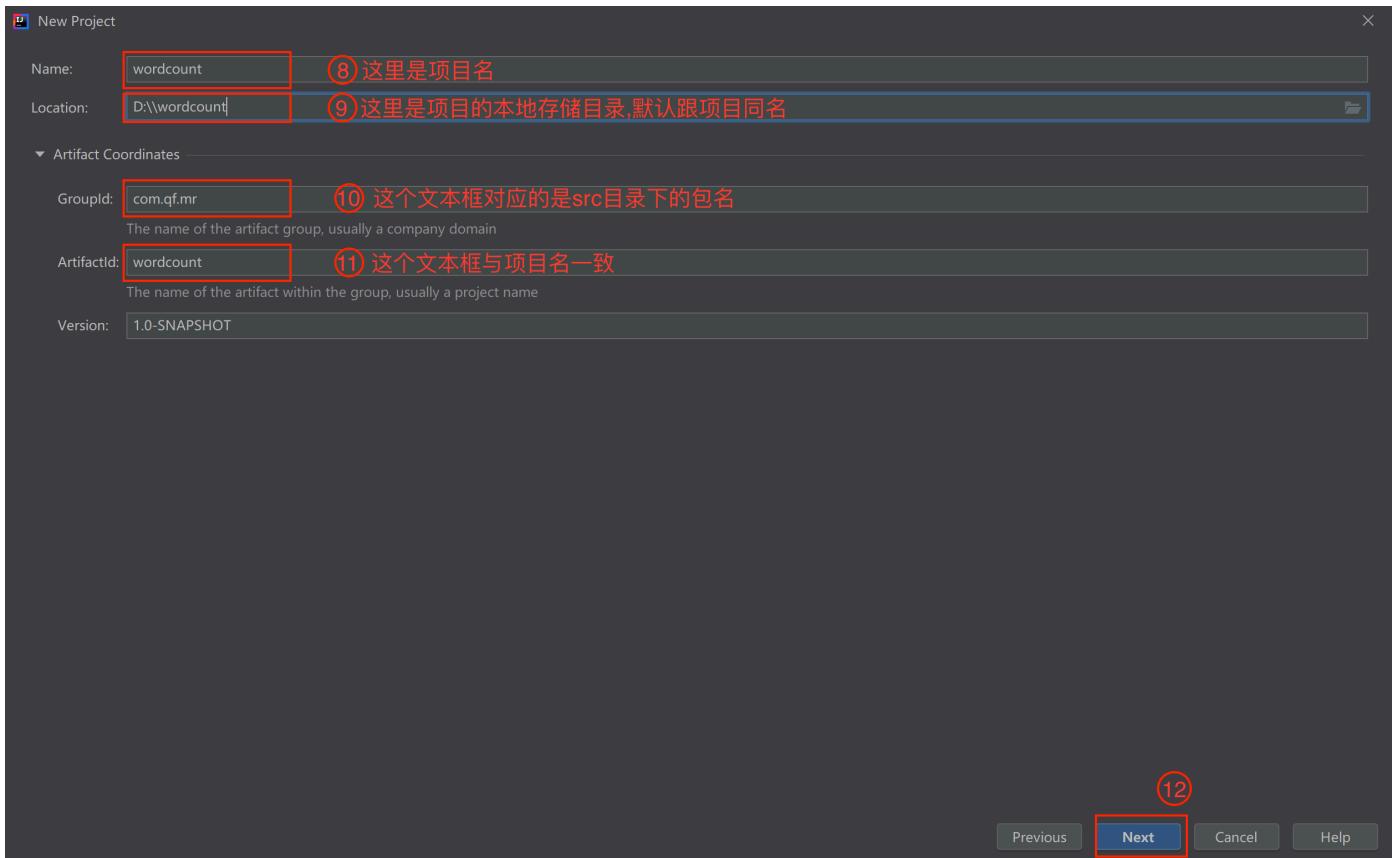
- 1 释义：
- 2 `create from archetype`, 选择quickstart
- 3 解释：这里的意思是直接快速创建一个maven项目的模板，包括初始的main方法, pom.xml文件的配置等都会自动设置好。再详细点说就是你创建完了就可以直接运行项目，打印我们的Hello World！
- 4
- 5 注意：如果不选择`create from archetype`, 直接创建项目，得到的是一个空架子，所有相关的环境都需要我们自己搭建(这种也经常使用)

参考下图中的1,2,3,4,5,6,7：

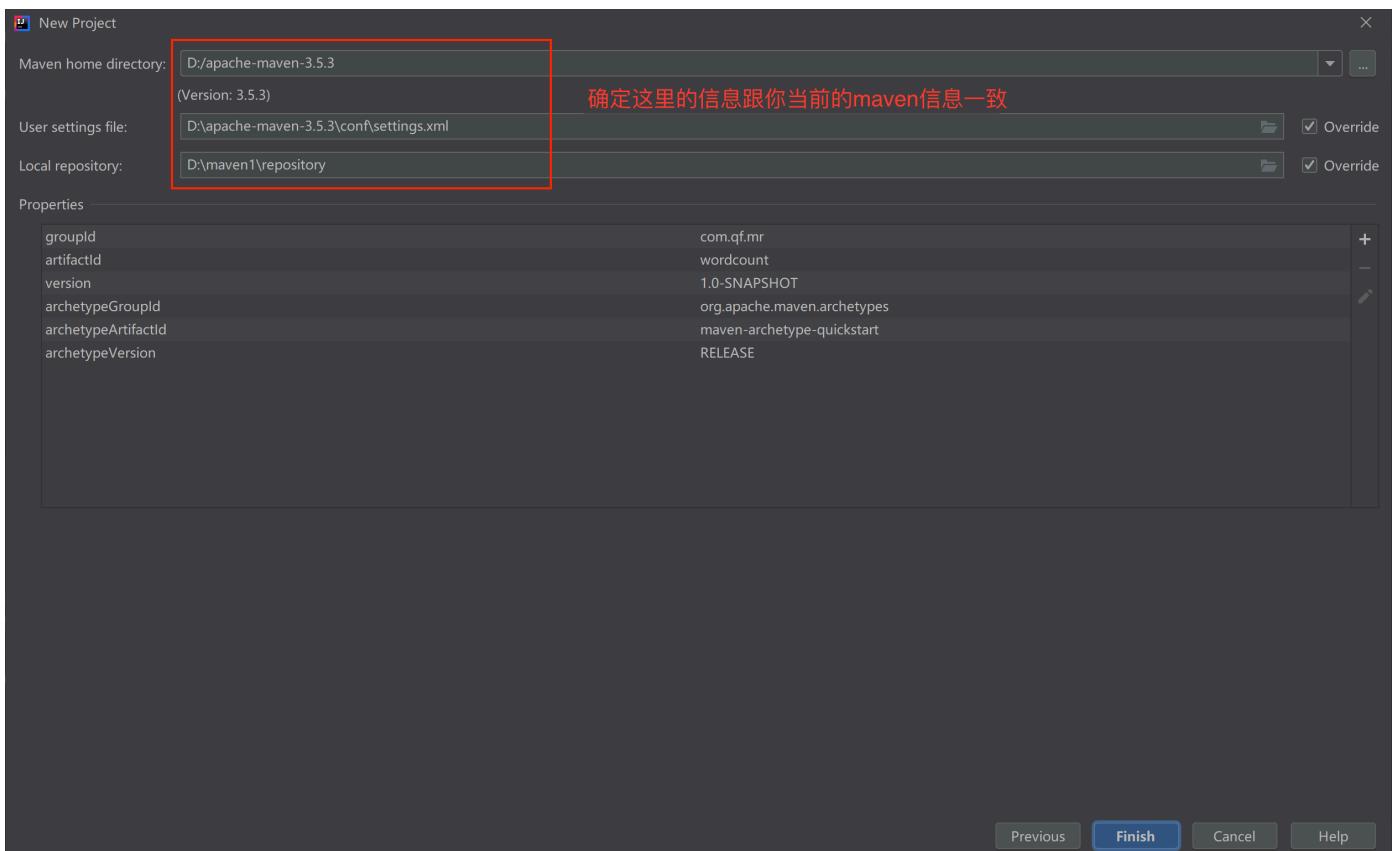


然后在弹出的新界面中的文本框中进行命名操作,参考下图中的8,9,10,11：

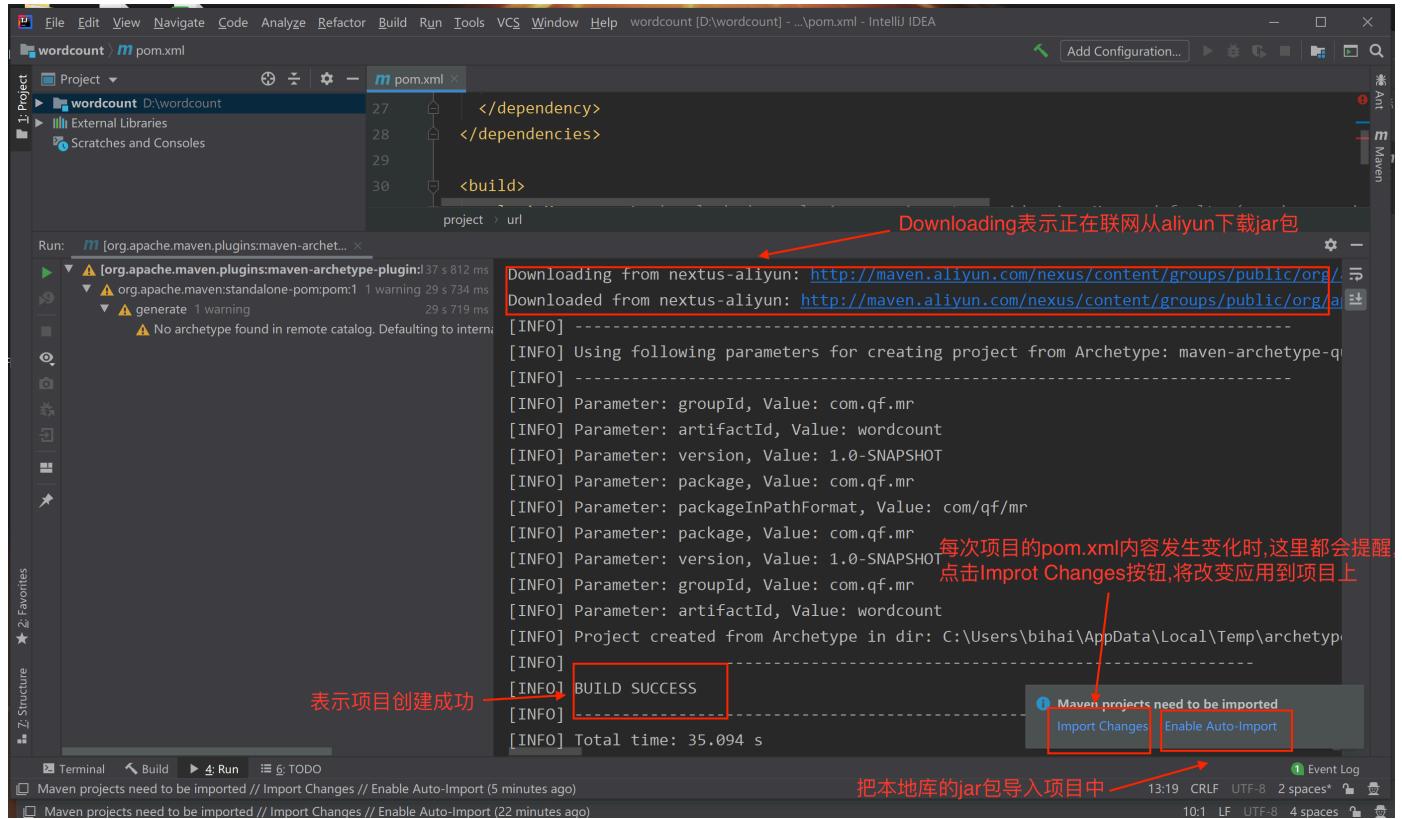
- 1 解析：
- 2
- 3 - `GroupId`: 这个文本框对应的是src目录下的包名
- 4
- 5 - `ArtifactId`: 这个文本框对应的是项目名



点击上一个图中的⑫ Next后, 来到如下窗口, 确定是不是你安装的maven, 如果不是, 选择一下你自己的maven路径。然后点击finish完成项目的创建。

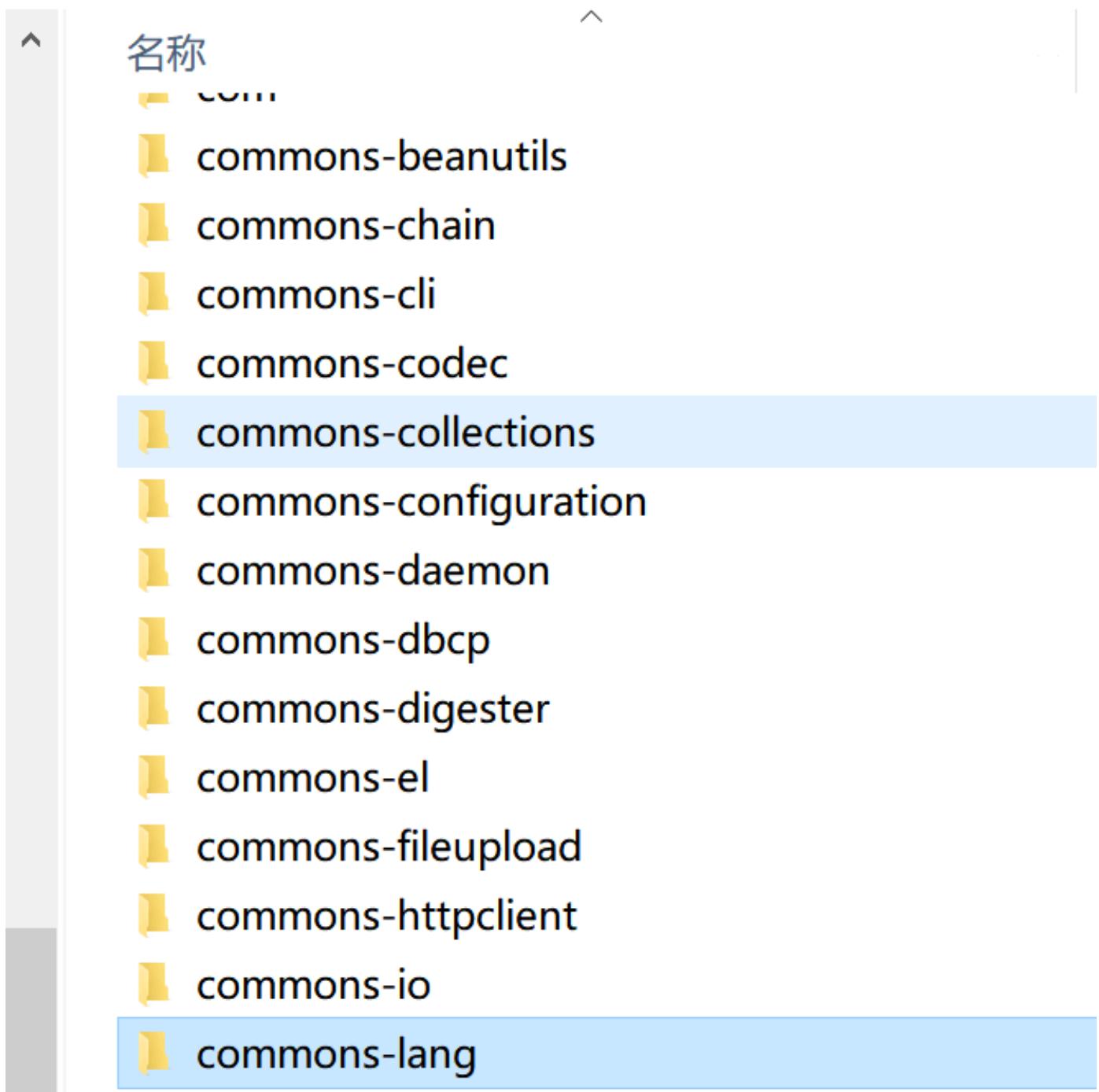


注意：当你刚刚创建完项目后，会出现如下界面，downloading表示正在从阿里云的远程仓库下载jar包到你的本地仓库中。第一次下载的时间可能会久一点。如果想要将jar包导入项目中，请点击右下角的Enable Auto-import



查看你的本地仓库目录，应该有内容了。例：如下图示

此电脑 > 新加卷 (D:) > maven1 > repository >



4.1.2.3. 运行Maven项目常见错误

3.1 初次使用Idea打包Maven项目报错 Non-parseable settings expected START_TAG or END_TAG

解决:在配置setting.xml时所有行要正确对齐.请将所有标签之间的空格删除掉, 然后一个一个Tab对齐, 于是惊喜来了。。。在Idea中Maven完美package成功! ! !

3.2 错误描述:'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 116, column 21

解决:版本号丢失

```
1 <build>
2     <plugins>
3         <plugin>
4
5             <groupId>org.apache.maven.plugins</groupId>
6                 <artifactId>maven-compiler-
7                     plugin</artifactId>
8                     <version>3.3</version> 把这里加上
9                     <configuration>
10                         <source>1.8</source>
11                         <target>1.8</target>
12                     </configuration>
13             </plugin>
14         </plugins>
15     </build>
```

4.1.3. 联网下载jar包导入本地Maven库(会)

4.1.3.1. 常用Maven仓库地址

```
1 https://mvnrepository.com/  
2  
3 https://maven.aliyun.com/mvn/search
```

4.1.3.2. 下载Hadoop的jar包到本地仓库

打开 <https://mvnrepository.com/> 网址，搜索hadoop

The screenshot shows the mvnrepository.com search interface. The URL in the address bar is `https://mvnrepository.com/search?q=hadoop`. A red arrow points from the text "地址" (Address) to the address bar. Another red arrow points from the text "搜索hadoop" (Search hadoop) to the search input field. The search results page displays "Found 2007 results". On the left, there's a sidebar titled "Repository" with links to Central (1.3k), Sonatype (563), Spring Plugins (320), Spring Lib M (253), and Talend (152). The main content area shows the first result: "1. Apache Hadoop Common" by org.apache.hadoop, specifically "hadoop-common".

先找到hadoop-common，点进去，找到相应版本号，进去查看maven的dependency信息。

This screenshot shows the detailed view for the "Apache Hadoop Common" artifact. It includes the artifact icon, name, group ID ("org.apache.hadoop"), artifact ID ("hadoop-common"), version ("2.7.6"), and a note about the last release on Sep 10, 2019. A red arrow points from the text "点进去" (Click here) to the artifact name "hadoop-common".

This screenshot shows the XML representation of the dependency. The URL in the address bar is `<!— https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common —>`. A red arrow points from the text "全选, Ctrl+C复制" (Select all, Ctrl+C copy) to the XML code. The code is as follows:

```
<dependency>  
    <groupId>org.apache.hadoop</groupId>  
    <artifactId>hadoop-common</artifactId>  
    <version>2.7.6</version>  
</dependency>
```

将dependency的内容，全选，复制到项目的pom.xml文件中的dependencies标签内部，如下图

```
<dependencies>←  
  <!-- https://mvnrepository.com/artifact/org.apache.h  
  <dependency>  
    <groupId>org.apache.hadoop</groupId>  
    <artifactId>hadoop-common</artifactId>  
    <version>2.7.6</version>←  
  </dependency>
```

然后，其他的jar包，如hadoop-client、hadoop-hdfs、hadoop-mapreduce-client-core这些jar的pom信息也依次复制到项目下的pom.xml文件中。

pom.xml内容如下：

```
1 <dependencies>  
2   <!--  
3     https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common-->  
4   <dependency>  
5     <groupId>org.apache.hadoop</groupId>  
6     <artifactId>hadoop-common</artifactId>  
7     <version>2.7.6</version>  
8   </dependency>  
9   <!--  
10    https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs-->  
11   <dependency>  
12     <groupId>org.apache.hadoop</groupId>  
13     <artifactId>hadoop-hdfs</artifactId>  
14     <version>2.7.6</version>  
15   </dependency>
```

```
14 <!--  
15   https://mvnrepository.com/artifact/org.apache.hadoop/ha  
16   doop-mapreduce-client-core -->  
17 <dependency>  
18   <groupId>org.apache.hadoop</groupId>  
19   <artifactId>hadoop-mapreduce-client-core</artifactId>  
20   <version>2.7.6</version>  
21 </dependency>  
22 <!--  
23   https://mvnrepository.com/artifact/org.apache.hadoop/ha  
24   doop-client-->  
25 <dependency>  
26   <groupId>org.apache.hadoop</groupId>  
27   <artifactId>hadoop-client</artifactId>  
28   <version>2.7.6</version>  
29 </dependency>  
30 </dependencies>
```

这样配置完，就会自动从远程仓库中下载这四个jar包，以及依赖的所有jar包到你的本地库了。

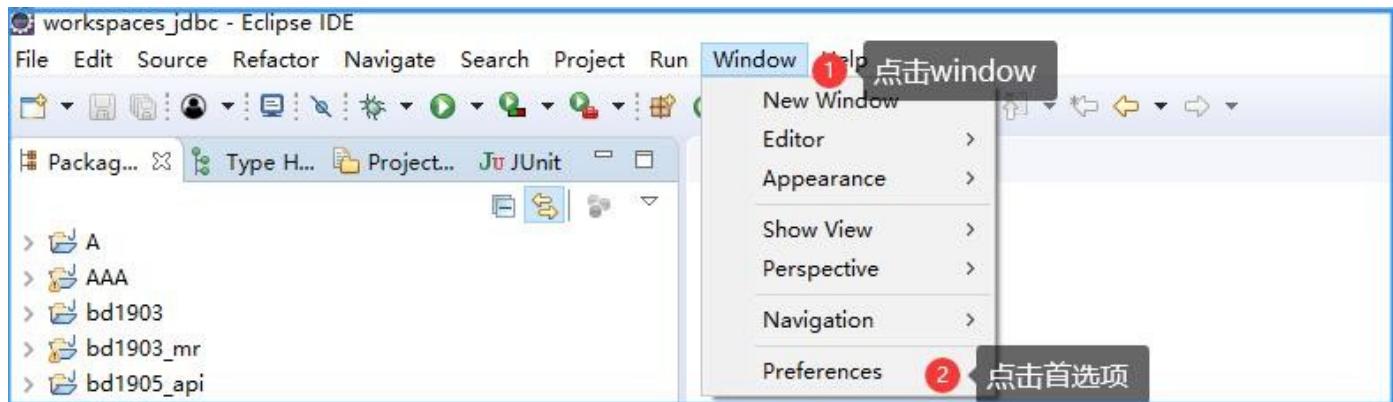
4.1.4. Eclipse与Maven工具的整合(了解)

4.1.4.1. 配置前说明

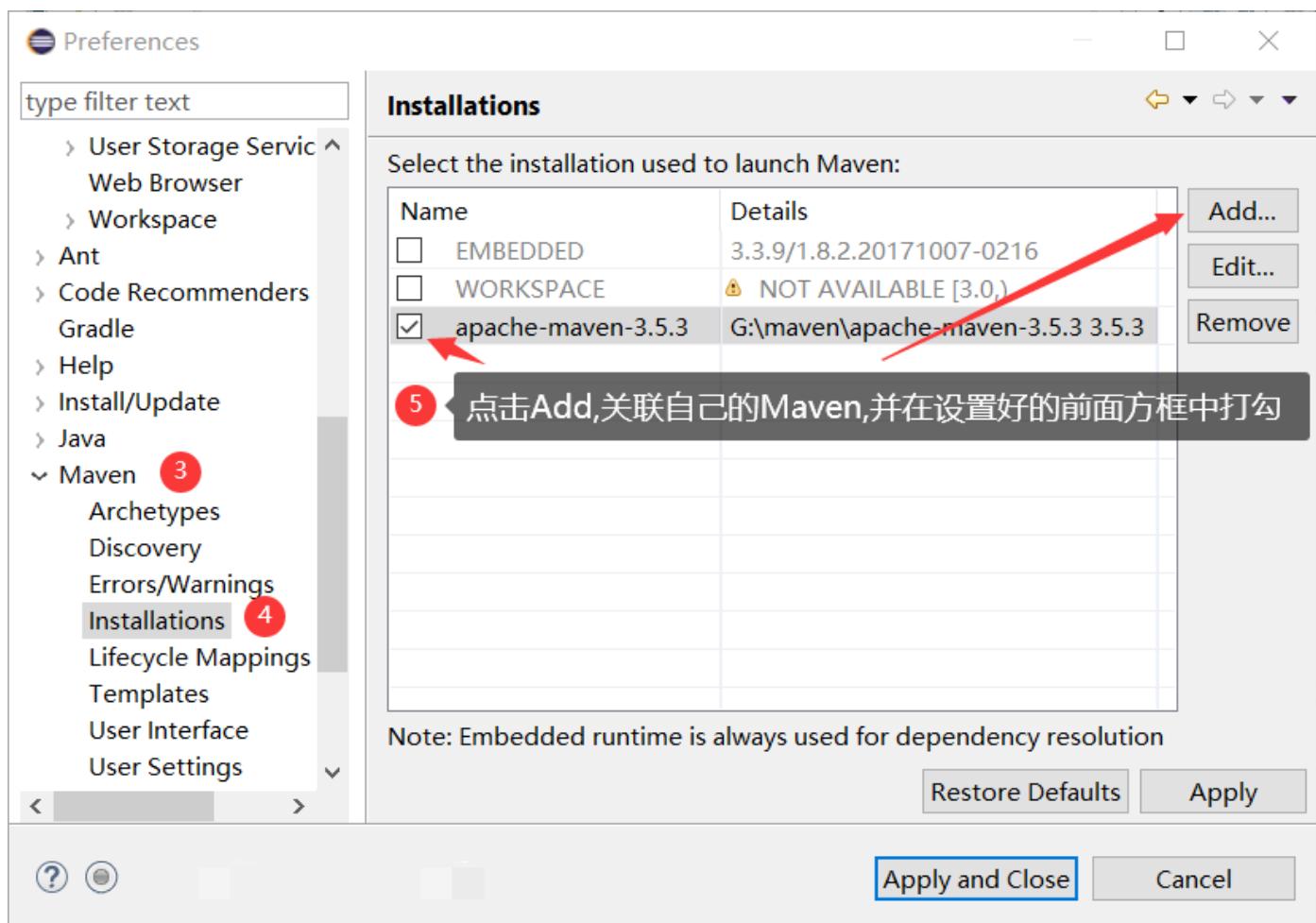
- 1 1. 如果你还没有安装maven安装包，那么请回到本文中的“第二节”内容进行安装、配置本地仓库、环境变量
- 2 2. 比较旧的eclipse版本，如果想与maven整合，那么需要先安装一个eclipse-maven插件，才能再整合。

4.1.4.2. Eclipse配置Maven

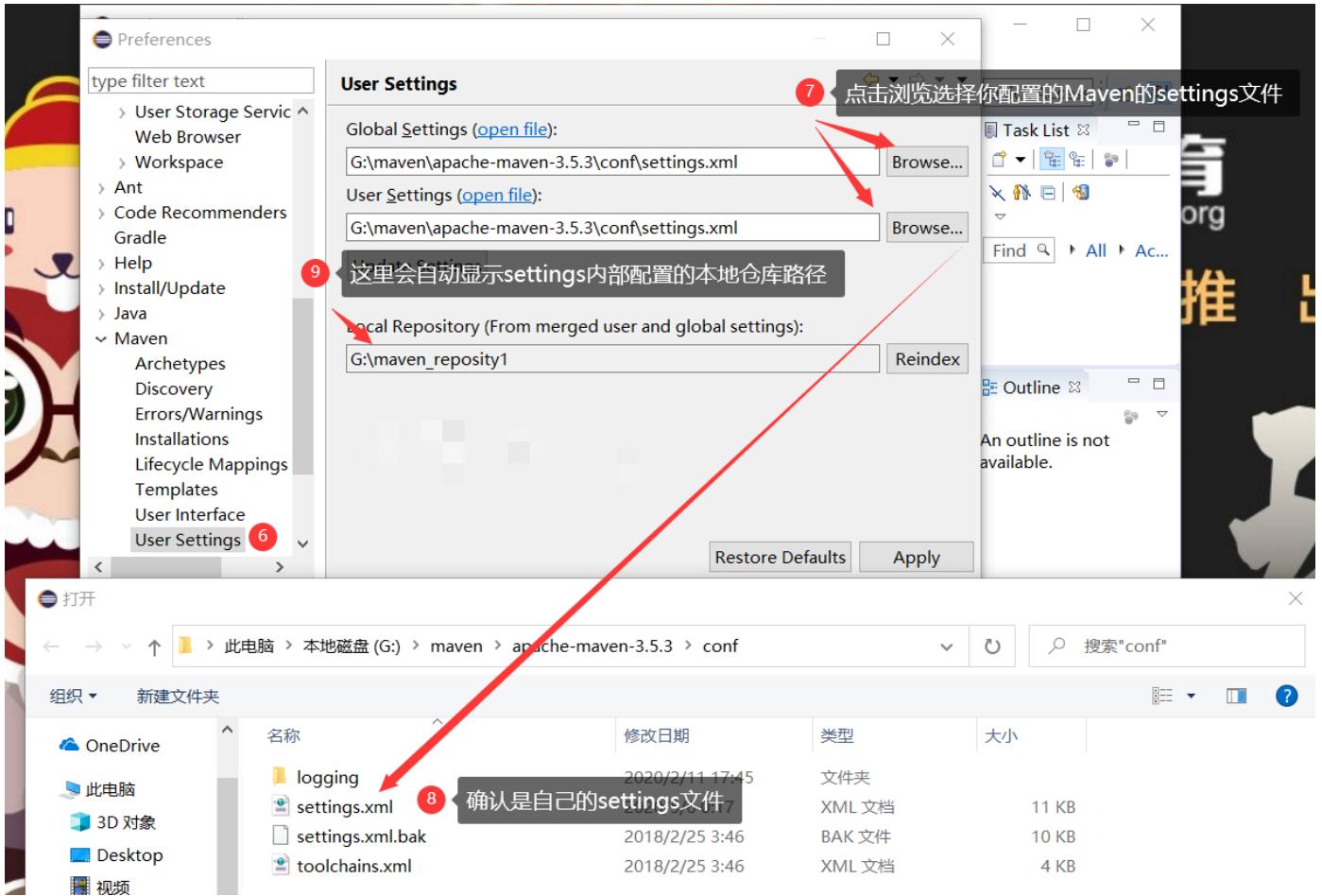
首先，选择window下的首选项，参考下图中的1,2步



然后,找到Maven中的Installations,选择Add,添加你安装的Maven,参考下图中的3,4,5



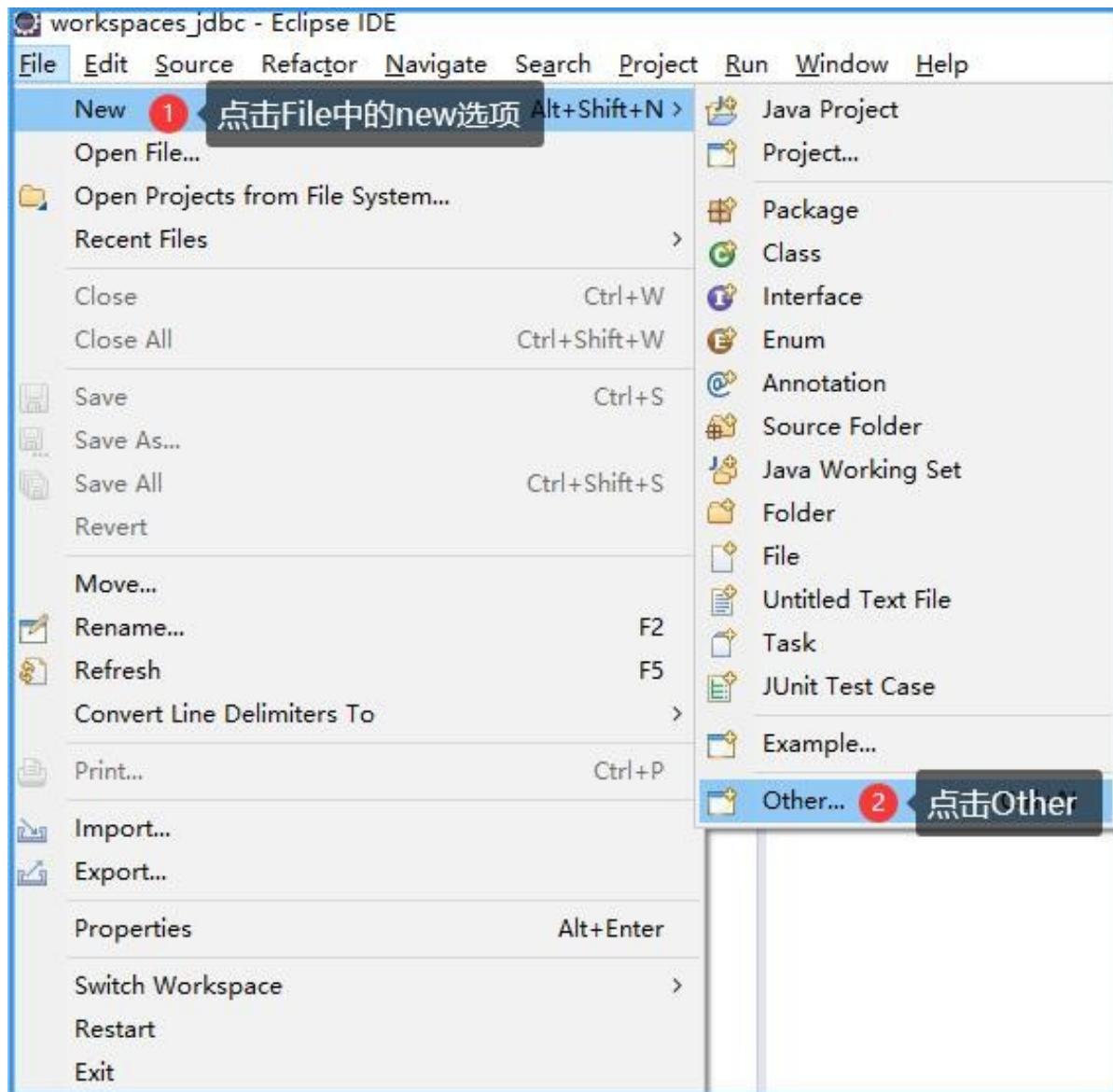
再选择Maven中的User Settings, 添加maven中的settings文件, 参考下图中的6,7,8,9



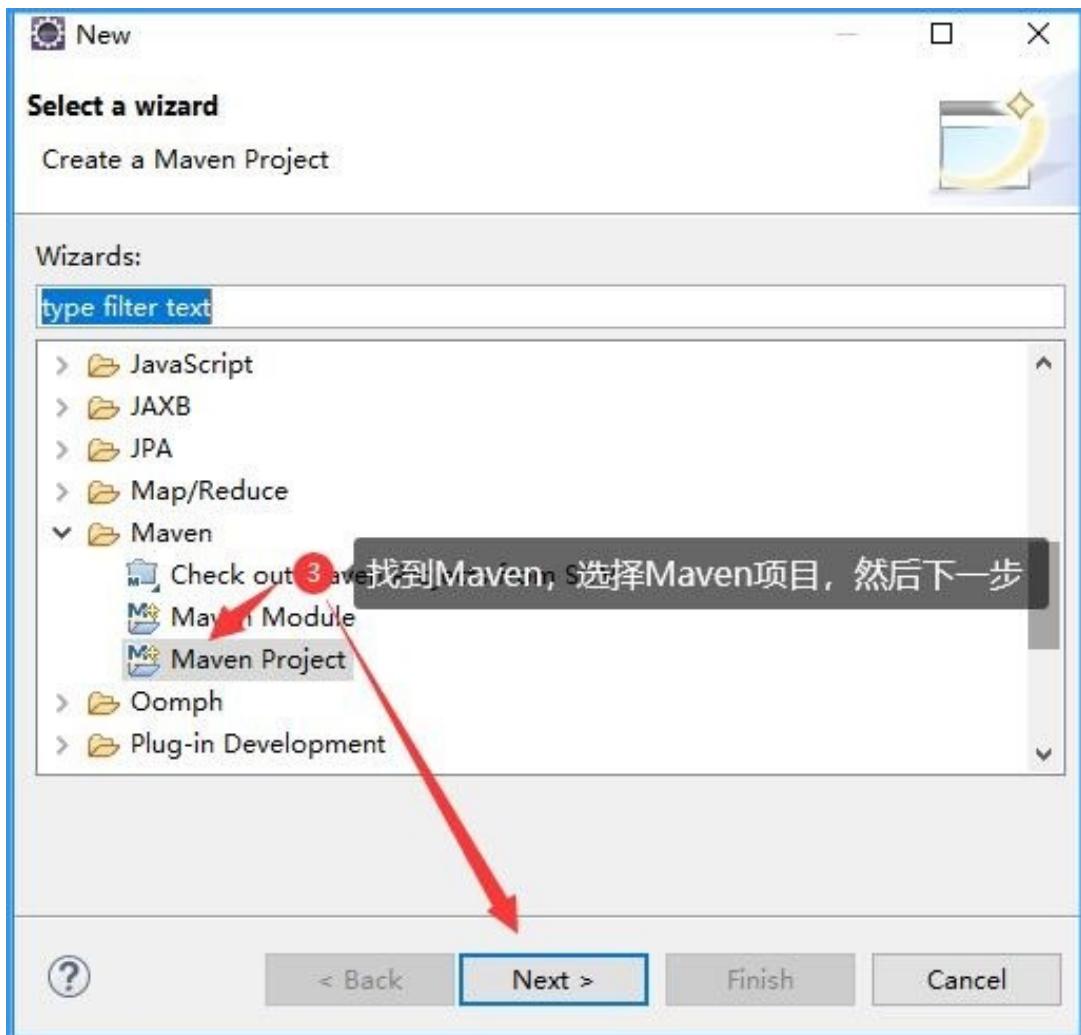
这样，eclipse就配置好了maven软件

4.1.4.3. 创建Maven项目

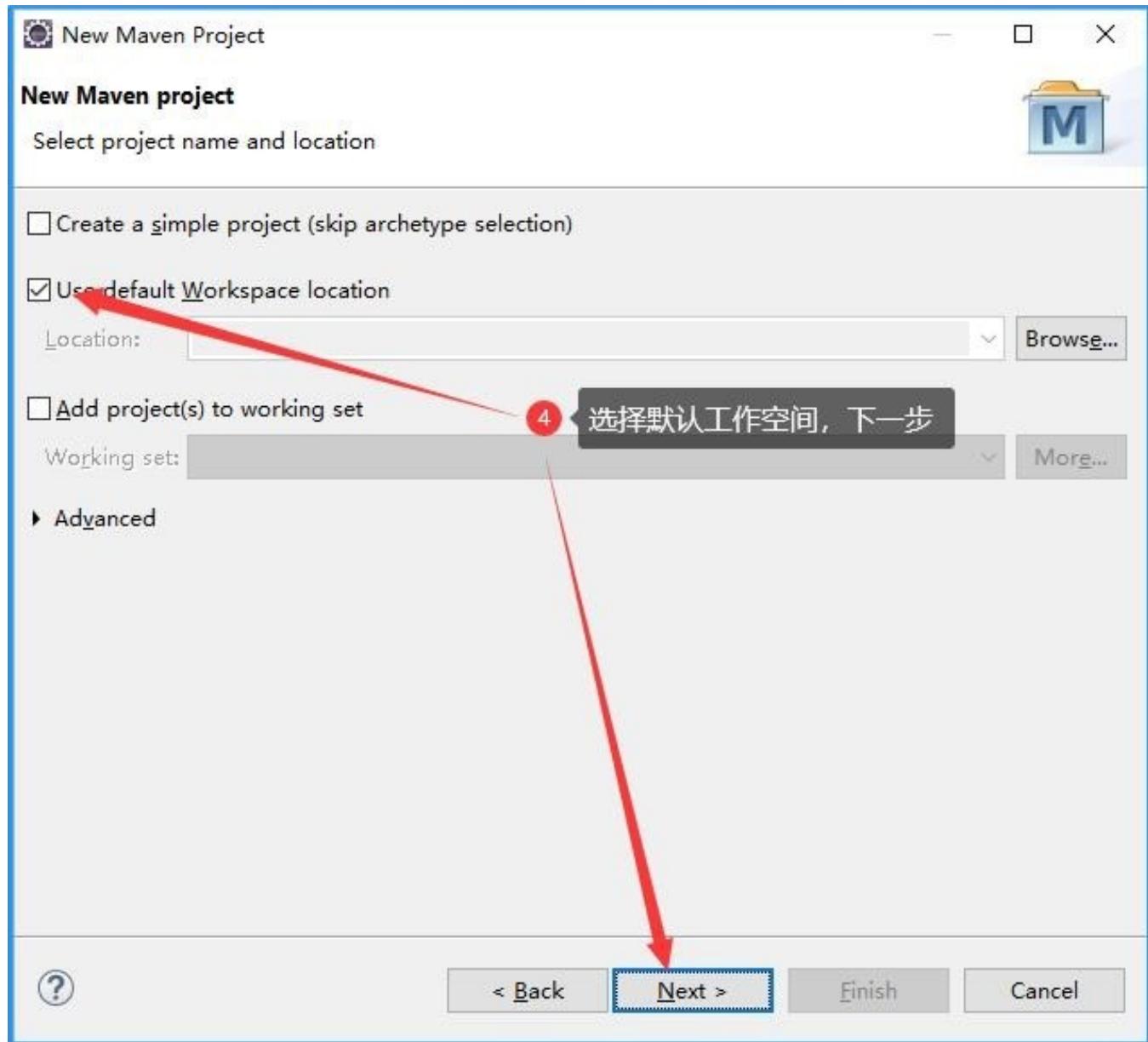
3.1 选择File中的New，去Other里寻找Maven，参考下图中的1，2。



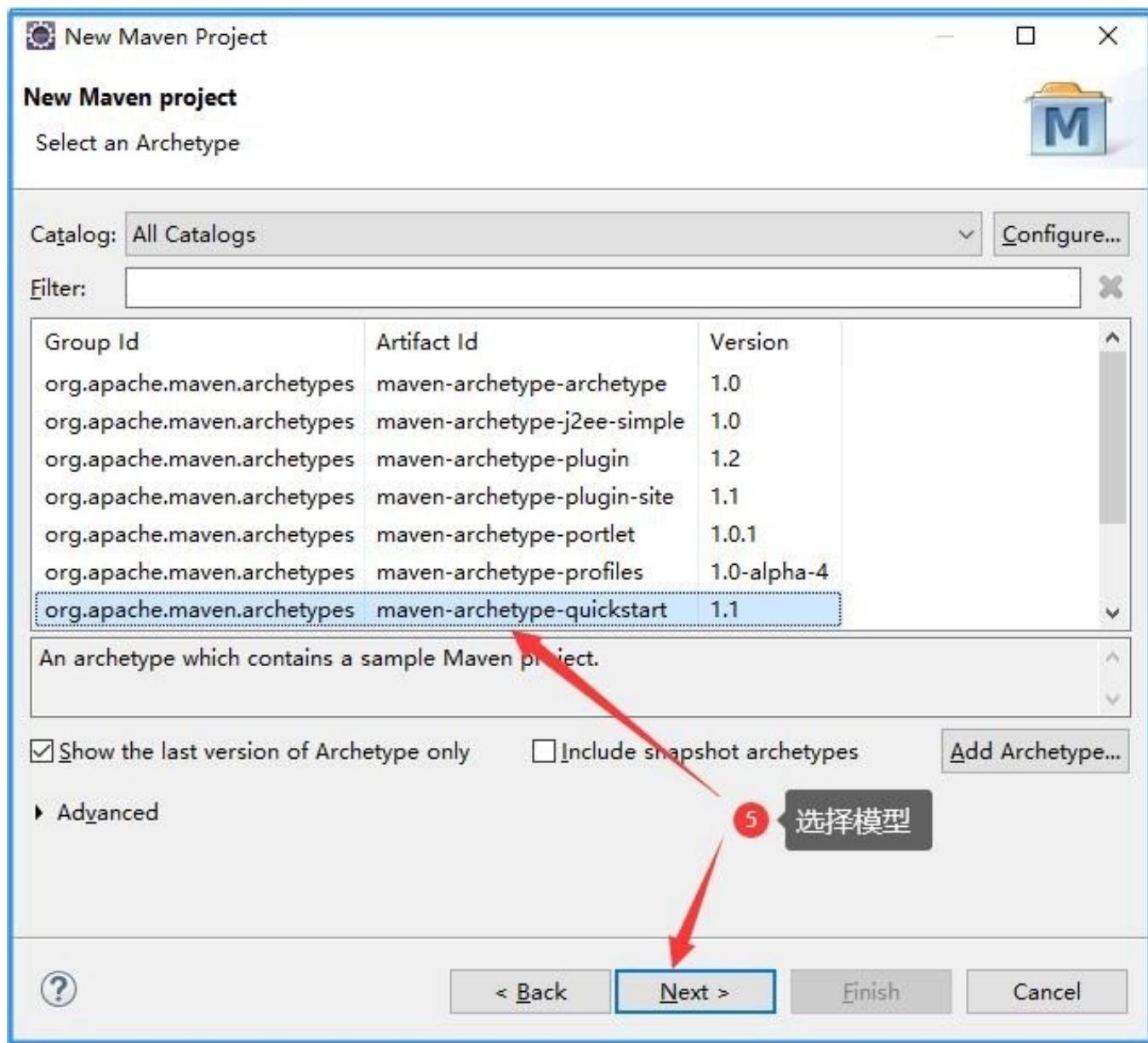
3.2 在Maven中选择Maven Project,进行创建, 参考下图中的3.



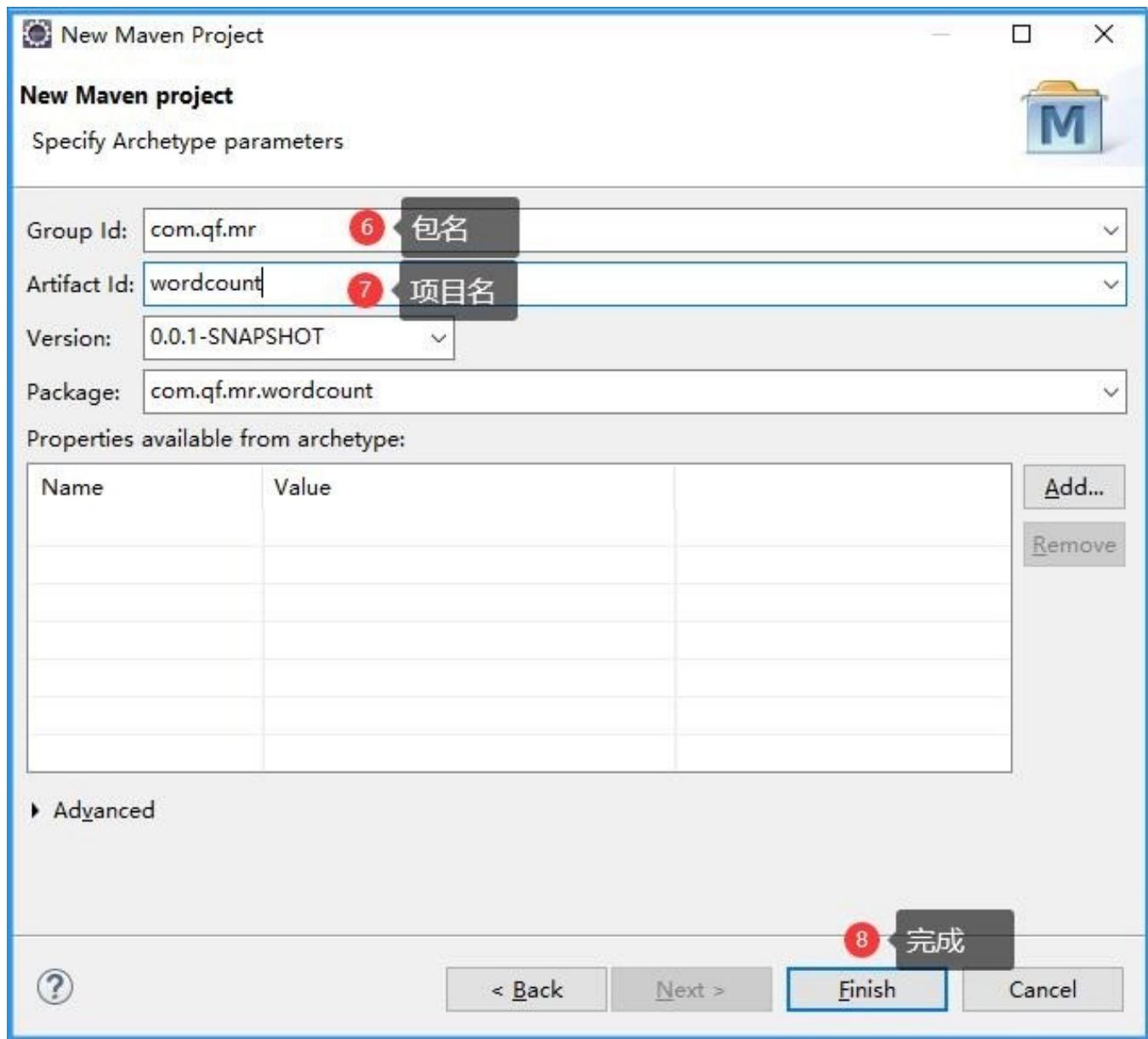
3.3 选择默认的工作空间，参考下图中的4.



3.4 然后选择创建maven项目的模型：quickstart，参考下图中的5.



3.5 在弹出的新窗口中的文本框中进行命名。参考下图中的6,7,8。



3.6 这样项目就创建好了，内部有一个pom.xml文件，用于配置项目依赖的jar文件信息

```

1<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3  <modelVersion>4.0.0</modelVersion>
4
5  <groupId>com.qf.mr</groupId>
6  <artifactId>wordcount</artifactId>
7  <version>0.0.1-SNAPSHOT</version>
8  <packaging>jar</packaging>
9
10 <name>wordcount</name>
11 <url>http://maven.apache.org</url>
12
13  <properties>
14      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15  </properties> 项目下有一个pom.xml文件，用于配置项目所需要的jar包信息
16
17  <dependencies>
18      <dependency>
19          <groupId>junit</groupId>
20          <artifactId>junit</artifactId>
21          <version>3.8.1</version>
22          <scope>test</scope>
23      </dependency>
24
25  </dependencies>
26 </project>
27
28

```

3.7 下面是开发mr所依赖的jar文件配置（如果是第一次，会进行联网下载，保证网络良好）

```

1 <dependency>
2   <groupId>jdk.tools</groupId>
3   <artifactId>jdk.tools</artifactId>
4   <version>1.6</version>
5   <scope>system</scope>
6   <systemPath>${JAVA_HOME}/lib/tools.jar</systemPath>
7 </dependency>
8 <!--
9 https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common -->
10 <dependency>
11   <groupId>org.apache.hadoop</groupId>
12   <artifactId>hadoop-common</artifactId>
13   <version>2.7.6</version>
14 </dependency>

```

```
14 <!--  
15   https://mvnrepository.com/artifact/org.apache.hadoop/ha  
16   doop-hdfs -->  
17 <dependency>  
18   <groupId>org.apache.hadoop</groupId>  
19   <artifactId>hadoop-hdfs</artifactId>  
20   <version>2.7.6</version>  
21 </dependency>  
22 <!--  
23   https://mvnrepository.com/artifact/org.apache.hadoop/ha  
24   doop-mapreduce-client-core -->  
25 <dependency>  
26   <groupId>org.apache.hadoop</groupId>  
27   <artifactId>hadoop-mapreduce-client-core</artifactId>  
28   <version>2.7.6</version>  
29 </dependency>  
30 <!--  
31   https://mvnrepository.com/artifact/org.apache.hadoop/ha  
32   doop-client -->  
33 <dependency>  
34   <groupId>org.apache.hadoop</groupId>  
35   <artifactId>hadoop-client</artifactId>  
36   <version>2.7.6</version>  
37 </dependency>
```

4.1.4.4 经常出现的问题总结

4.1 问题总结1

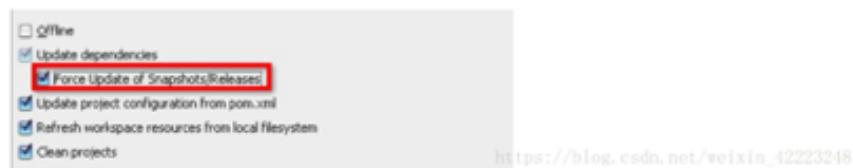
问题描述: maven下载安装,并配置到tomcat产生Could not calculate build plan: Plugin org.apache.maven.plugins:maven-resource

解决方法：

解决方法如下：

(1) 将C:\User\user.m2\repository\org\apache\maven\plugins\maven-deploy-plugin目录下的文件夹删除，

(2) 然后右击项目Maven->Update Project->Update Dependencies (如果更新无效，选择强制更新如下图)



如果还是不行，可以打开并修改conf/settings.xml,添加如下内容：

```
1 <!-- 设置本地仓库位置-->
2 <localRepository>G:\maven1\repository</localRepository>
3
4 <!-- 设置远程仓库-->
5 <mirror>
6   <id>nexus-aliyun</id>
7   <mirrorOf>*</mirrorOf>
8   <name>Nexus aliyun</name>
9   <url>http://maven.aliyun.com/nexus/content/groups/public
</url>
9 </mirror>
```

注：

- 1.本地仓库在eclipse中默认是 C:\Users\user.m2\repository 可自定义，注意配置路径不能出现中文
2. mirror表示的是访问镜像。如果本地仓库没有项目所需要的jar包，就会通过这里的mirror配置的url地址进行从远程仓库获取需要的jar，同时将这个jar添加到本地目录中，当再次使用的时候，就会直接从本地仓库中直接获取。我这里选择的是阿里云

4.2 问题总结2

解决: 包(hadoopjdk,javajdk,mavenjdk)的安装路径不能出现中文,空格(比如:Program files)

4.3 问题总结3

软件的版本问题,有时版本会出现不匹配,更改相关的版本即可

4.4 问题总结4

问题描述: 遇到Missing artifact jdk.tools:jar:1.8的问题

原因: tools.jar包是JDK自带的,pom.xml中依赖的包隐式依赖tools.jar包,而tools.jar并未在库中,只需将tools.jar包添加到jdk库中即可

解决方案: 在pom文件中添加如下代码

```
1 <dependency>
2   <groupId>com.sun</groupId>
3   <artifactId>tools</artifactId>
4   <version>1.8.0</version>
5   <scope>system</scope>
6   <systemPath>${env.JAVA_HOME}/lib/tools.jar</systemPath>
7 </dependency>
```

默认pom.xml如果不指定JDK版本,版本为1.5,而有些项目需要使用泛型等特性,所以需要使用1.8版本的JDK,要手动修改pom.xml

涉及到的插件还有maven-compiler-plugin

打开pom.xml,增加配置节点

```
<!—局部jdk配置,pom.xml中>
```

```
1 <build>
2     <plugins>
3         <plugin>
4
5             <groupId>org.apache.maven.plugins</groupId>
6                 <artifactId>maven-compiler-
7                     plugin</artifactId>
8                         <configuration>
9                             <source>1.8</source>
10                            <target>1.8</target>
11                        </configuration>
12                    </plugin>
13                </plugins>
14            </build>
```

4.5 问题总结5

问题描述: 配置完maven,运行@Test代码出现下面的错误

```
1 java.lang.NoSuchMethodError:  
2   org.junit.runner.Request.classWithoutSuiteMethod(Ljava/  
3     lang/Class;)Lorg/junit/runner/Request;  
4  
5 at  
6   org.eclipse.jdt.internal.junit4.runner.JUnit4TestLoader  
7     .createFilteredTest(JUnit4TestLoader.java:79)  
8  
9 at  
10  org.eclipse.jdt.internal.junit4.runner.JUnit4TestLoader  
11    .createTest(JUnit4TestLoader.java:71)  
12  
13 at  
14  org.eclipse.jdt.internal.junit4.runner.JUnit4TestLoader  
15    .loadTests(JUnit4TestLoader.java:46)  
16  
17 at  
18  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.  
19    runTests(RemoteTestRunner.java:522)  
20  
21 at  
22  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.  
23    runTests(RemoteTestRunner.java:760)  
24  
25 at  
26  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.  
27    main(RemoteTestRunner.java:206)
```

原因: junit的版本低了,使用的是junit 4.10,改成4.12即可

4.6 问题总结6

问题描述: 原来的项目的更新成现在的maven库

解决: 右击当前的项目---选项:Maven---选项:Update Project—在出现的页面中选择要更新的工程,点击更新即可

4.1.5. Maven配置时pom.xml文件的说明(会)

4.1.5.1 一个pom.xml文件展示

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.qf.mr</groupId>
8   <artifactId>wordcount</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <!-- 项目名字 -->
11  <name>wordcount</name>
12  <!-- FIXME change it to the project's website -->
13  <url>http://www.example.com</url>
14
15  <!-- 基本常用变量定义 -->
16  <properties>
17    <project.build.sourceEncoding>UTF-
18  </project.build.sourceEncoding>
19    <!-- hadoop 包版本 -->
20    <hadoop.version>2.7.1</hadoop.version>
21    <!-- log4j日志文件管理包版本 -->
22    <slf4j.version>1.6.6</slf4j.version>
```

```
21      <log4j.version>1.2.9</log4j.version>
22  </properties>
23  <dependencies>
24      <!-- junit所依赖jar包 -->
25      <dependency>
26          <groupId>junit</groupId>
27          <artifactId>junit</artifactId>
28          <version>4.10</version>
29          <scope>test</scope>
30      </dependency>
31      <!-- jdk.tools 依赖jar包 -->
32      <dependency>
33          <groupId>jdk.tools</groupId>
34          <artifactId>jdk.tools</artifactId>
35          <version>1.7</version>
36          <scope>system</scope>
37
38      <systemPath>${JAVA_HOME}/lib/tools.jar</systemPath>
39      </dependency>
40      <!-- hadoop-common 依赖jar包 -->
41      <dependency>
42          <groupId>org.apache.hadoop</groupId>
43          <artifactId>hadoop-common</artifactId>
44          <version>${hadoop.version}</version>
45          <scope>provided</scope>
46      </dependency>
47      <!-- hadoop-hdfs 依赖jar包 -->
48      <dependency>
49          <groupId>org.apache.hadoop</groupId>
50          <artifactId>hadoop-hdfs</artifactId>
51          <version>${hadoop.version}</version>
52          <scope>provided</scope>
53      </dependency>
```

```
53      <!-- 日志文件管理包 -->
54      <!-- log start -->
55      <dependency>
56          <groupId>log4j</groupId>
57          <artifactId>log4j</artifactId>
58          <version>${log4j.version}</version>
59      </dependency>
60      <dependency>
61          <groupId>org.slf4j</groupId>
62          <artifactId>slf4j-api</artifactId>
63          <version>${slf4j.version}</version>
64      </dependency>
65      <dependency>
66          <groupId>org.slf4j</groupId>
67          <artifactId>slf4j-log4j12</artifactId>
68          <version>${slf4j.version}</version>
69      </dependency>
70      <!-- log end -->
71  </dependencies>
72  <!-- maven构建的环境设置 -->
73  <build>
74      <!-- 插件管理 -->
75          <pluginManagement><!-- lock down plugins versions
76          to avoid using Maven defaults (may be moved to parent
77          pom) -->
78              <!-- 插件列表 -->
79              <plugins>
80                  <plugin>
81                      <artifactId>maven-project-info-reports-
82                      plugin</artifactId>
83                          <version>3.0.0</version>
84                      </plugin>
85                  </plugins>
```

```
83      </pluginManagement>
84    </build>
85  </project>
```

4.1.5.2. pom.xml常用元素介绍

Maven中的project 包含的一些约束信息

```
1 modelVersion : 指定当前pom的版本
2 groupId : (主项目标示, 定义当前maven属于哪个项目, 反写公司网址
+项目名)
3 artifactId : (实际项目模块标识, 项目名+模块名)
4 version (当前项目版本号, 第一个0标识大版本号, 第二个0标识分支版
本号, 第三个0标识小版本号, 0.0.1, snapshot快照, alpha内部测
试, beta公测, release稳定, GA正式发布)
5 properties : 一些常用属性的声明(包括依赖的版本号, 编码, 资源位
置等)
6 name : 项目描述名
7 url : 项目地址
8 description : 项目描述
9 developers : 开发人员列表
10 licenses : 许可证
11 organization: 组织
12 dependencies: 依赖列表
13 dependency: 依赖项目 里面放置坐标
14 scope: 包的依赖范围
15 optional : 设置依赖是否可选
16 exclusions: 排除依赖传递列表
17 dependencyManagement : 依赖的管理
18 build: 为构建行为提供支持
19 plugins: 插件列表
20 parent: 子模块对父模块的继承
21 modules: 聚合多个maven项目
```

4.1.5.3. 对scope包依赖范围的说明

3.1. 依赖范围

maven中三种classpath:编译, 测试, 运行 1.compile: 默认范围, 编译测试运行都有效 2.provided: 在编译和测试时有效 3.runtime: 在测试和运行时有效 4.test : 只在测试时有效 5.system : 在编译和测试时有效, 与本机系统关联, 可移植性差

3.2. 注意范围

问题描述:maven scope 'provided' 和 'compile'的区别 2.1. 解释 其实这个问题很简单。对于scope=compile的情况 (默认scope),也就是说这个项目在编译, 测试, 运行阶段都需要这个artifact(模块)对应的jar包在classpath中。而对于scope=provided的情况, 则可以认为这个provided是目标容器已经provide这个artifact。换句话说, 它只影响到编译, 测试阶段。在编译测试阶段, 我们需要这个artifact对应的jar包在classpath中, 而在运行阶段, 假定目标的容器 (比如我们这里的liferay容器) 已经提供了这个jar包, 所以无需我们这个artifact对应的jar包了。

2.2. 实例 (scope=provided) 比如说, 假定我们自己的项目ProjectABC中有一个类叫C1,而这个C1中会import这个portal-impl的artifact中的类B1, 那么在编译阶段, 我们肯定需要这个B1, 否则C1通不过编译, 因为我们的scope设置为provided了, 所以编译阶段起作用, 所以C1正确的通过了编译。测试阶段类似, 故忽略。那么最后我们要吧ProjectABC部署到Liferay服务器上了, 这时候, 我们到\$liferay-tomcat-home\weapps\ROOT\WEB-INF\lib下发现, 里面已经有了一个portal-impl.jar了, 换句话说, 容器已经提供了这个artifact对应的jar,所以, 我们在运行阶段, 这个C1类直接可以用容器提供的portal-impl.jar中的B1类, 而不会出任何问题。

2.3. 实际插件的行为 刚才我们讲述的是理论部分，现在我们看下，实际插件在运行时候，是如何来区别对待scope=compile和scope=provided的情况的。做一个实验就可以很容易发现，当我们用maven install生成最终的构件包ProjectABC.war后，在其下的WEB-INF/lib中，会包含我们被标注为scope=compile的构件的jar包，而不会包含我们被标注为scope=provided的构件的jar包。这也避免了此类构件当部署到目标容器后产生包依赖冲突。