

课程目标

1. Maven介绍

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

2. Maven 安装配置

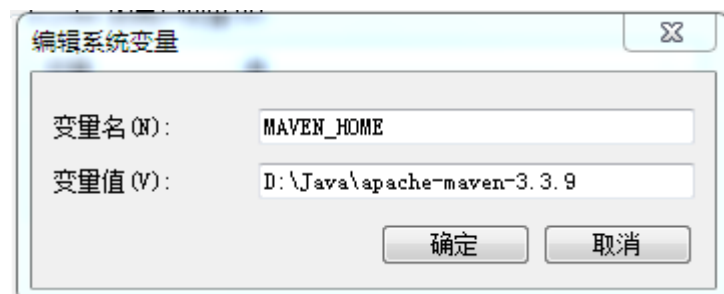
2.1Maven的安装

第一步：安装jdk，要求1.6或以上版本。

第二步：把maven解压缩，解压目录最好不要有中文。



第三步：配置环境变量MAVEN_HOME



第四步：配置环境变量PATH，将%MAVEN_HOME%\bin加入Path中，在Windows中一定要注意要用分号；与其他值隔开。

第五步：验证是否安装成功，打开cmd窗口，输入mvn -v

```
管理员: C:\Windows\system32\CMD.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>mvn -v
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-11T00:41:47+08:00)
Maven home: D:\Java\apache-maven-3.3.9\bin\..
Java version: 1.7.0_80, vendor: Oracle Corporation
Java home: D:\Java\jdk1.7.0_80\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"

C:\Users\Administrator>
```

2.2 Maven的配置

Maven有两个settings.xml配置文件，一个是全局配置文件，一个是用户配置文件。

全局配置

) > Java > apache-maven-3.3.9 > conf >		
名称	修改日期	类型
logging	2018/5/8 9:21	文件夹
settings.xml	2015/11/10 11:38	XML 文档
toolchains.xml	2015/11/10 11:38	XML 文档

%MAVEN_HOME%/conf/settings.xml 是maven全局的配置文件。

该配置文件中配置了本地仓库的路径，默认就是：~/.m2/repository。

其中~表示当前用户路径C:\Users\[UserName]。

> 用户 > JIKE > .m2 >	
名称	修改日期
repository	2018/5/

localRepository：用户仓库，用于检索依赖包路径

用户配置


~/.m2/settings.xml是用户的配置文件（默认没有该文件，需要将全局配置文件拷贝过来在进行修改）

注意：一般本地仓库的地址不使用默认配置，通常情况下需要在用户配置中，配置新的仓库地址。

配置步骤如下：

第一步：创建一个本地仓库目录，比如D:\java\repository\maven_repository。

第二步：复制maven的全局配置文件到~/.m2目录下，即创建用户配置文件

本地磁盘 (C:) > 用户 > JIKE > .m2		
	名称	修改日期
	 settings.xml	2018/5/8 9:35

第三步：修改maven的用户配置文件。

```
5
6 <!-- 本地仓库地址 -->
7 <localRepository>D:\java\repository\maven_repository</localRepository>
8 <mirrors>
9   <mirror>
10     <id>nexus-aliyun</id>
11     <mirrorOf>*</mirrorOf>
12     <name>Nexus aliyun</name>
13     <url>http://maven.aliyun.com/nexus/content/groups/public</url>
14   </mirror>
15 </mirrors>
```

完整配置：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
5             http://maven.apache.org/xsd/settings-1.0.0.xsd">
6   <mirrors>
7     <mirror>
8       <id>nexus-aliyun</id>
9       <mirrorOf>*</mirrorOf>
10      <name>Nexus aliyun</name>
11      <url>http://maven.aliyun.com/nexus/content/groups/public</url>
12    </mirror>
13  </mirrors>
14  <profiles>
15    <profile>
16      <id>jdk-1.8</id>
17      <activation>
18        <activeByDefault>true</activeByDefault>
19        <jdk>1.8</jdk>
20      </activation>
21      <properties>
22        <maven.compiler.source>1.8</maven.compiler.source>
23        <maven.compiler.target>1.8</maven.compiler.target>
24        <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
25      </properties>
26    </profile>
27  </profiles>
28 </settings>
```

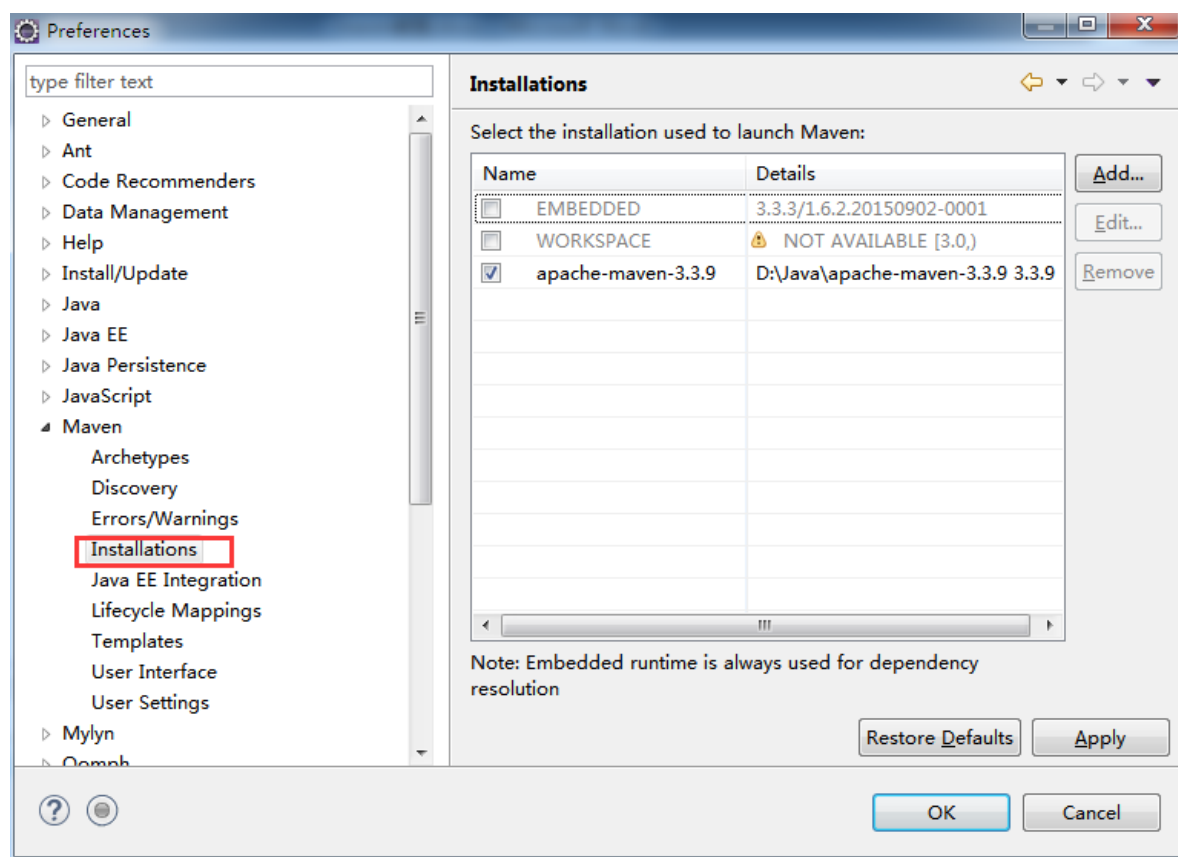
3.Maven 工程结构

1	项目名称	
2	src	
3	main	
4	java	存放项目的.java文件
5	resources	存放项目配置文件jdbc.properties
6	test	
7	java	存放所有测试.java文件，如JUnit测试类
8	resources	测试资源文件
9	target	目标文件输出位置例如.class、.jar、.war文件
10	pom.xml	maven项目核心配置文件

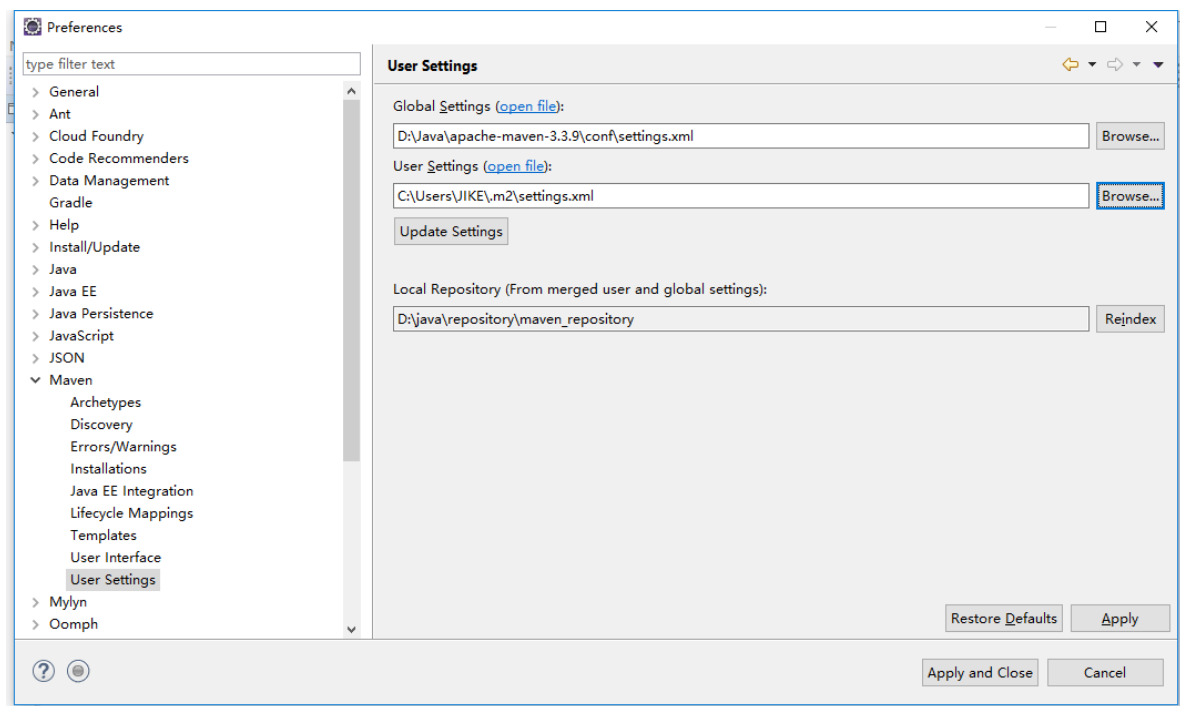
4. Eclipse 集成

M2Eclipse是eclipse中的maven插件，新版本的eclipse已经集成了该插件，可以直接使用

4.1 设置maven安装目录



4.2 用户配置

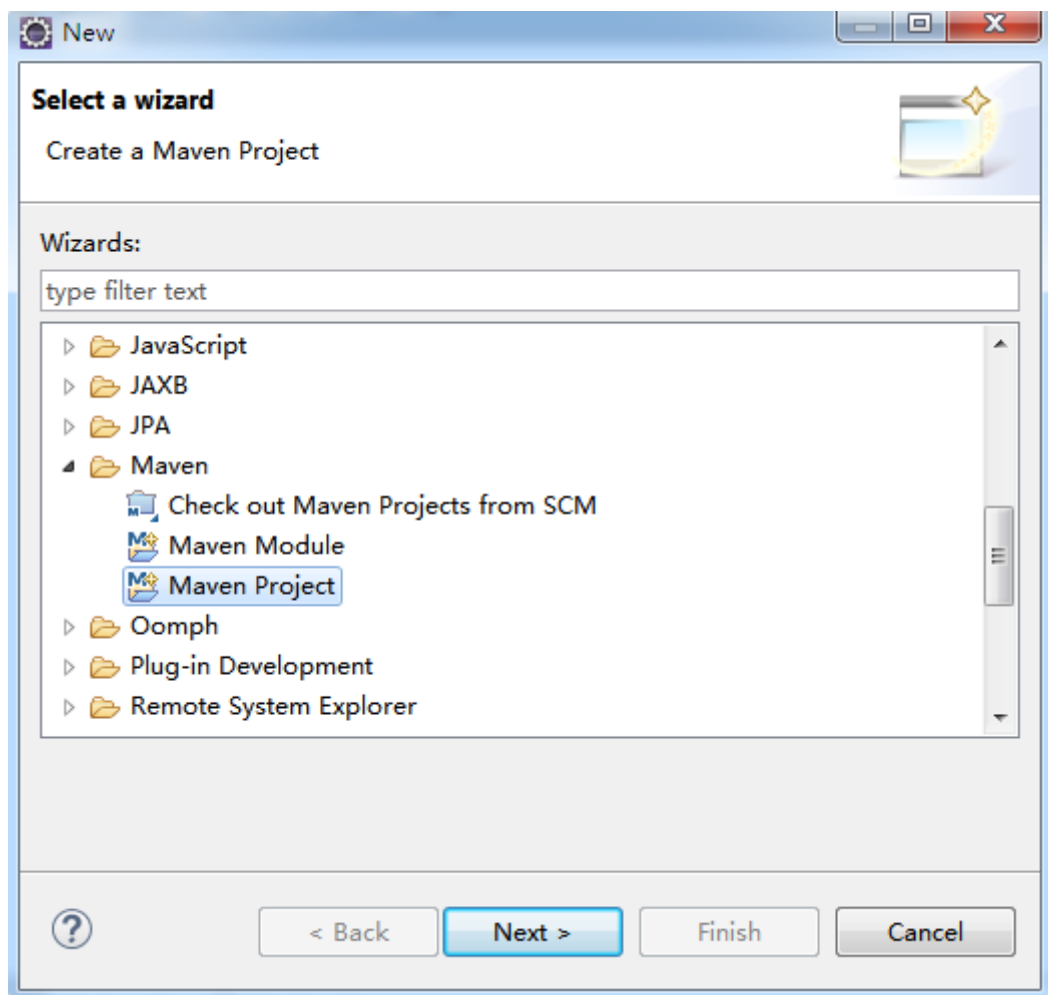


4.3 创建maven工程

通过骨架创建maven工程

创建工程

第一步：选择new→maven→Maven Project



第二步: next

New Maven Project

New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

☒ Use default Workspace location

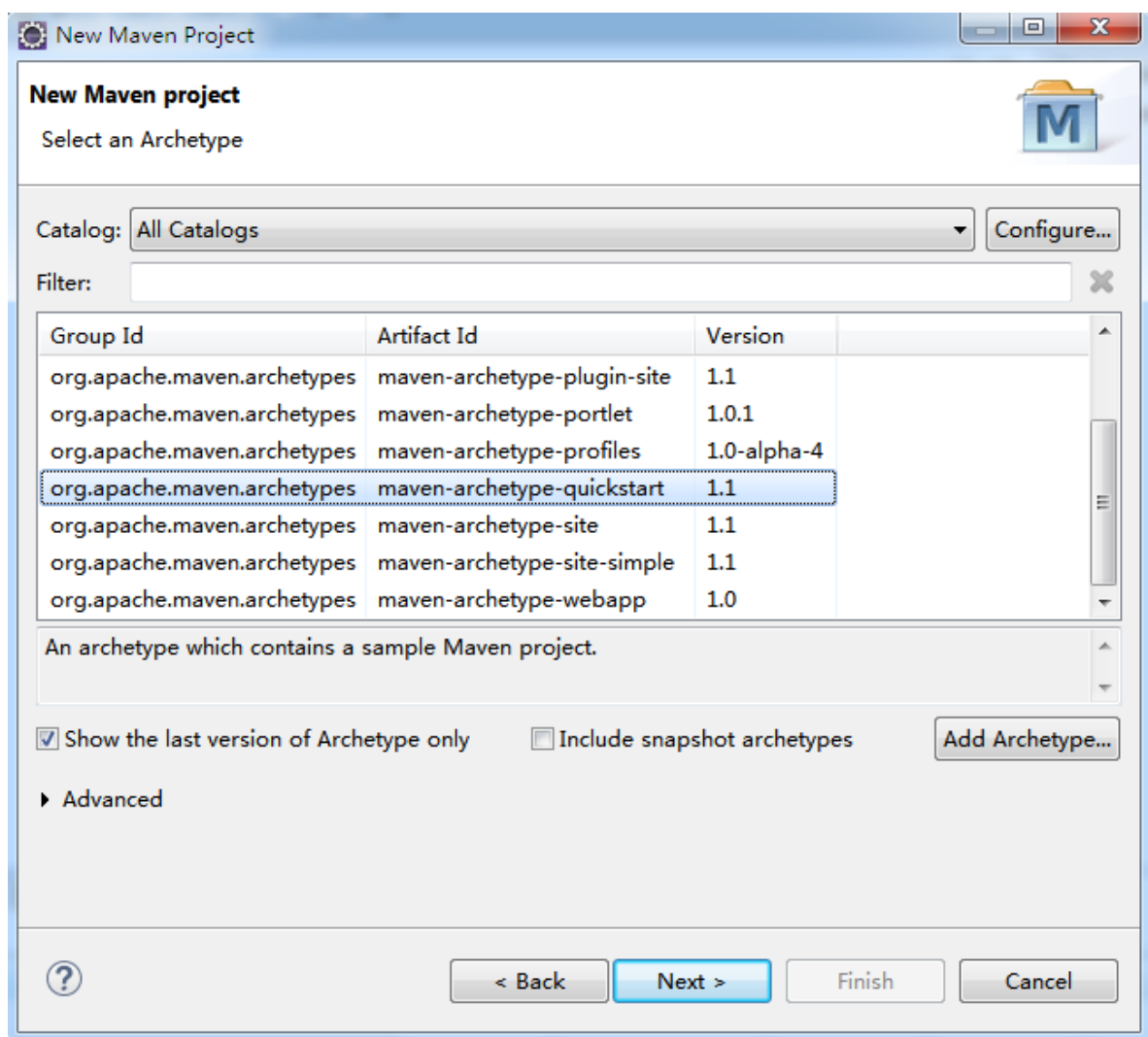
Location:

☐ Add project(s) to working set

Working set:

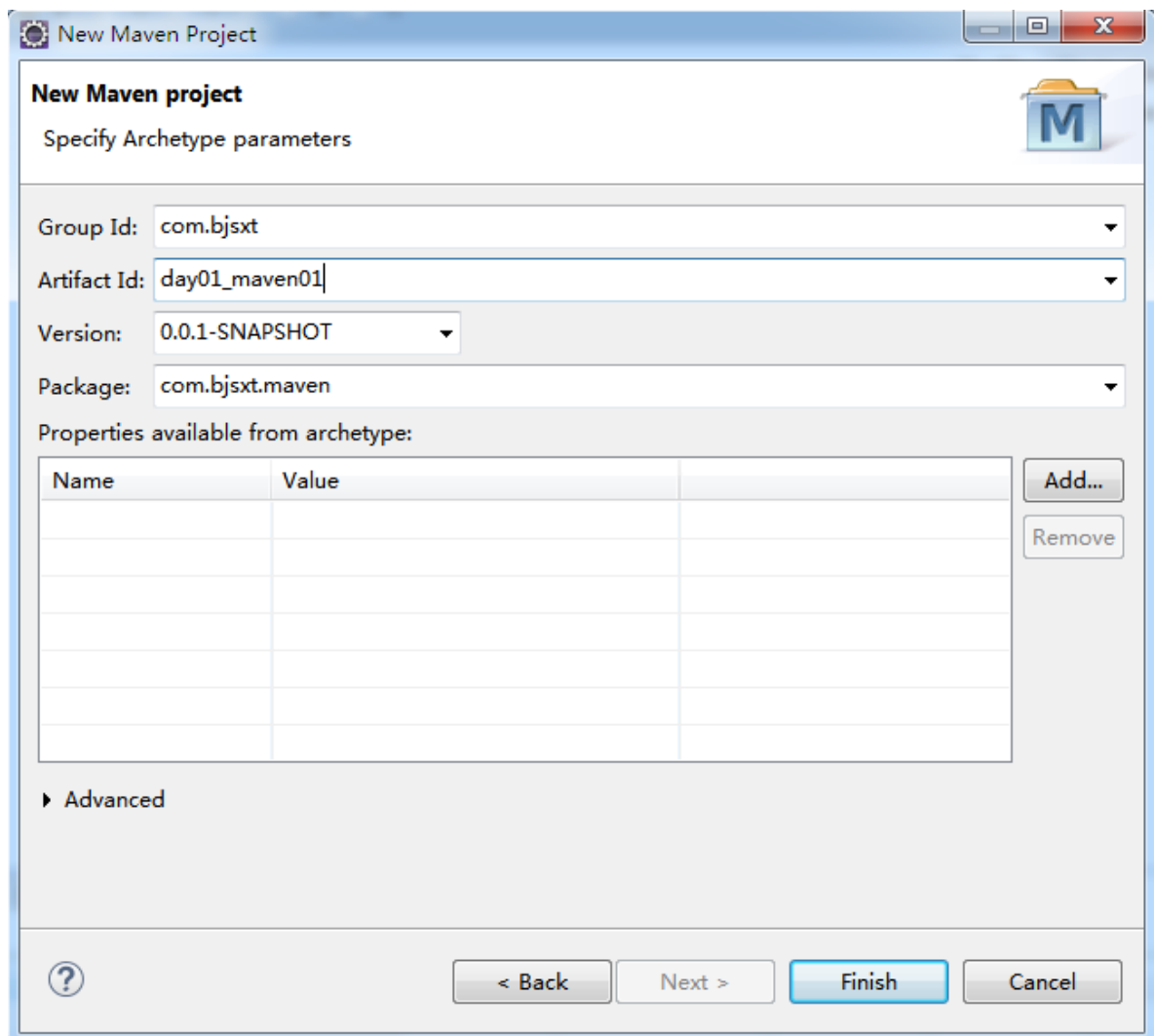
► Advanced

第三步: next

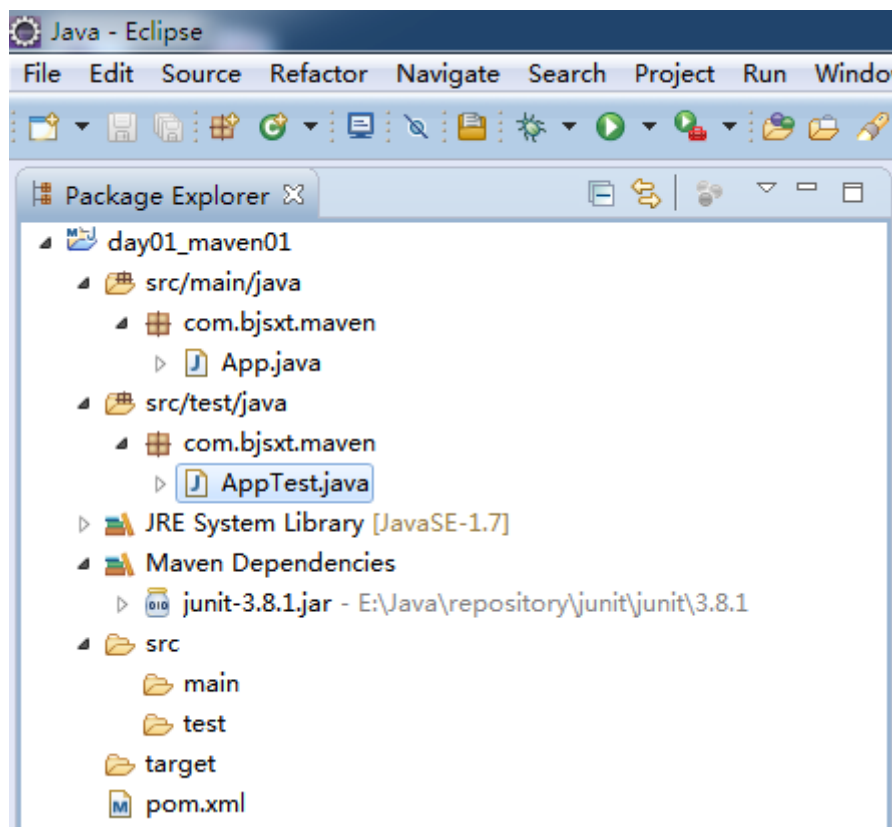


选择maven的工程骨架，这里我们选择quickstart。

第四步：next

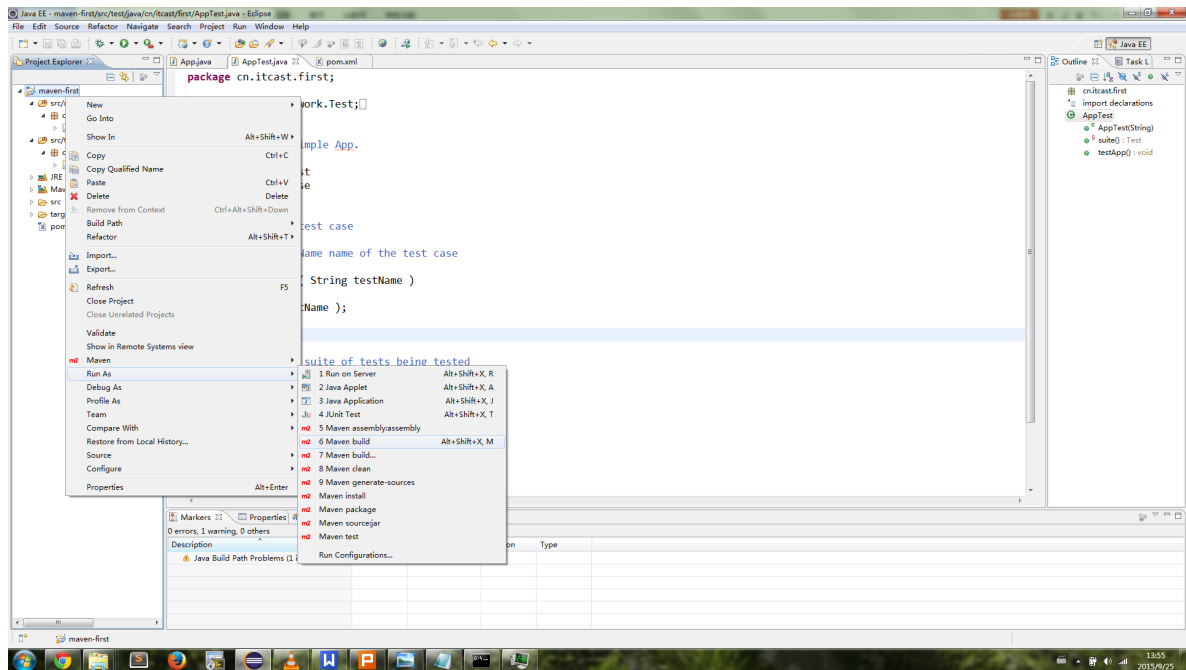


输入GroupId（组织名称）、ArtifactId（项目名称）、Version（版本号）、Package（包名）信息点击finish完成。



执行maven命令进行测试

在Eclipse的maven插件中执行maven命令，需要在maven工程或者pom.xml文件上点击右键，选择Run as→maven build..



可以在菜单中看到maven常用的命令已经以菜单的形式出现。

例如：

Maven clean

Maven install

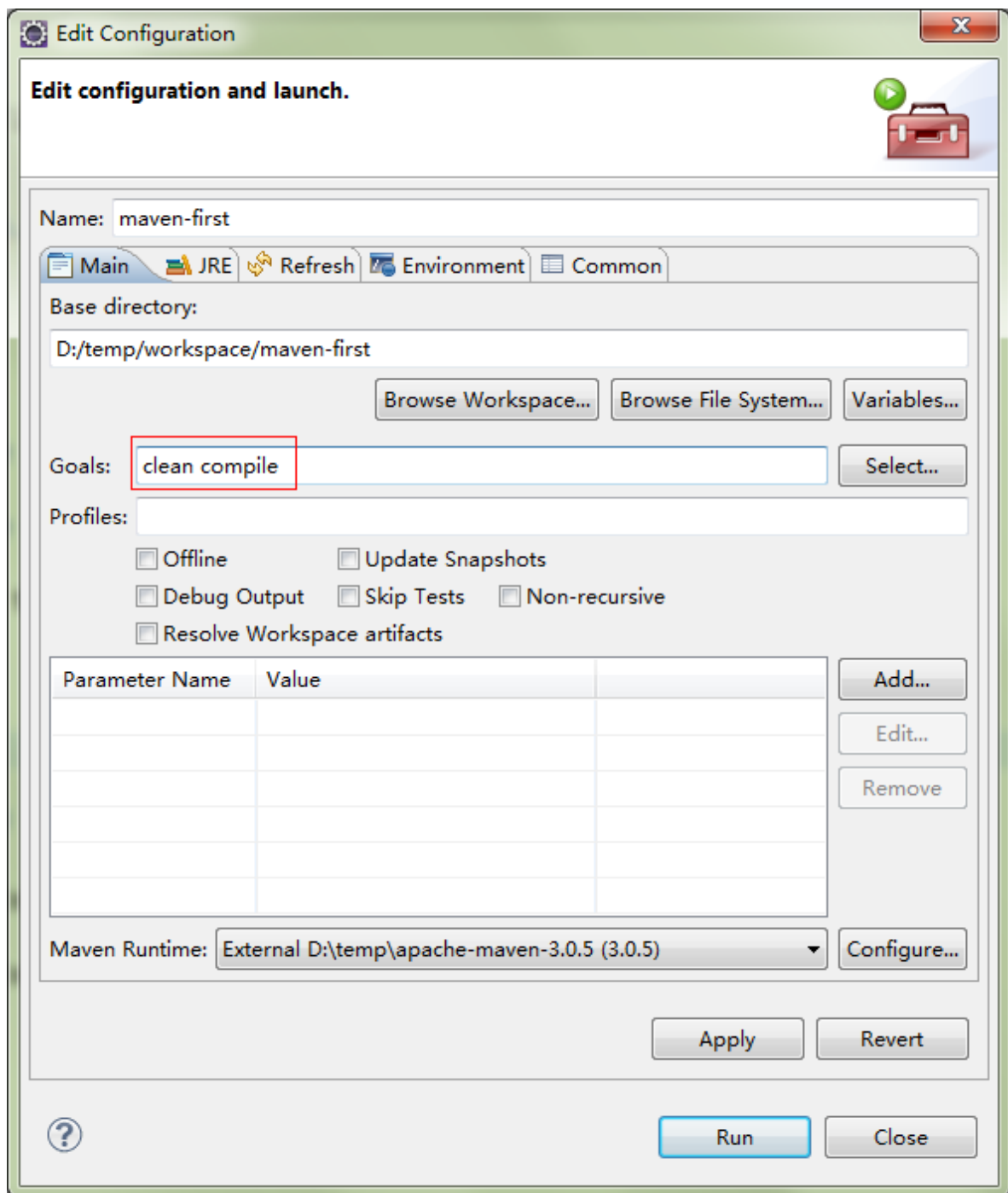
Maven package

Maven test

Maven build和maven build... 并不是maven的命令。

maven build...只是提供一个命令输入功能，可以在此功能中输入自定义的maven命令。

maven build的功能就是执行上次自定义命令。

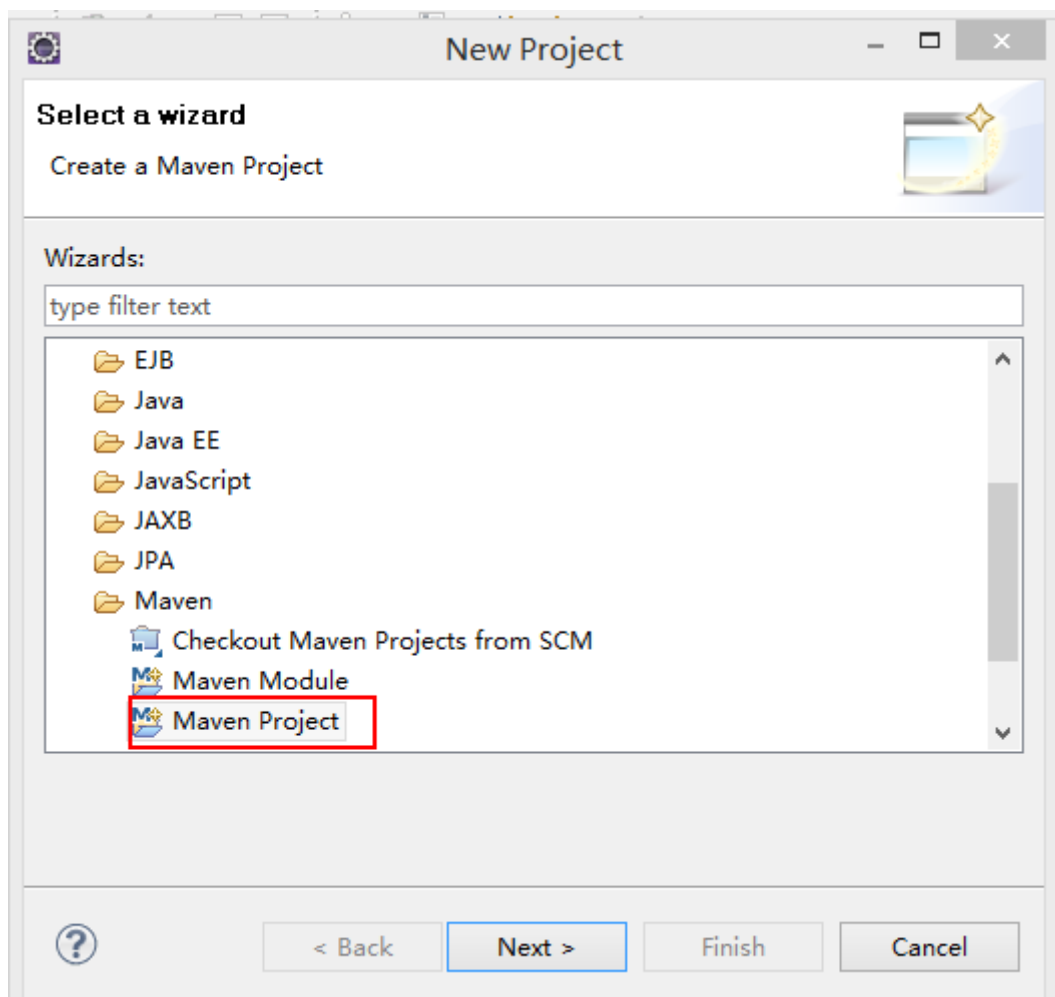


不通过骨架创建maven工程

通过选择骨架创建maven工程，每次选择骨架时都需要联网下载，如果网络不通或者较慢的情况下会有很长时间的等待。使用很是不方便，所以创建工程时可以不选择骨架直接创建工程。

创建工程

第一步：选择new→maven→Maven Project



第二步: next

New Maven Project

New Maven project

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

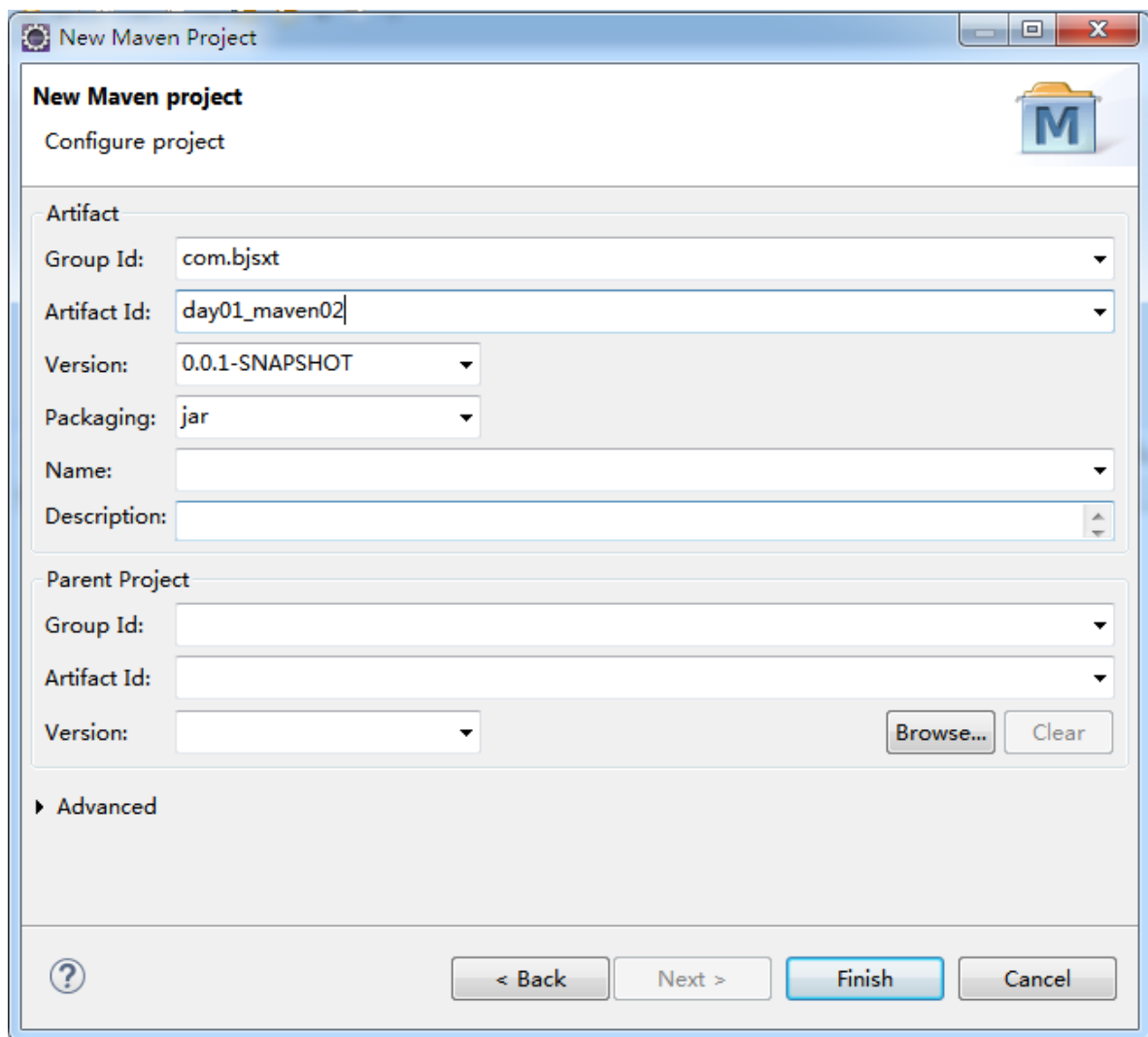
Location:

☐ Add project(s) to working set

Working set:

► Advanced

第三步：next

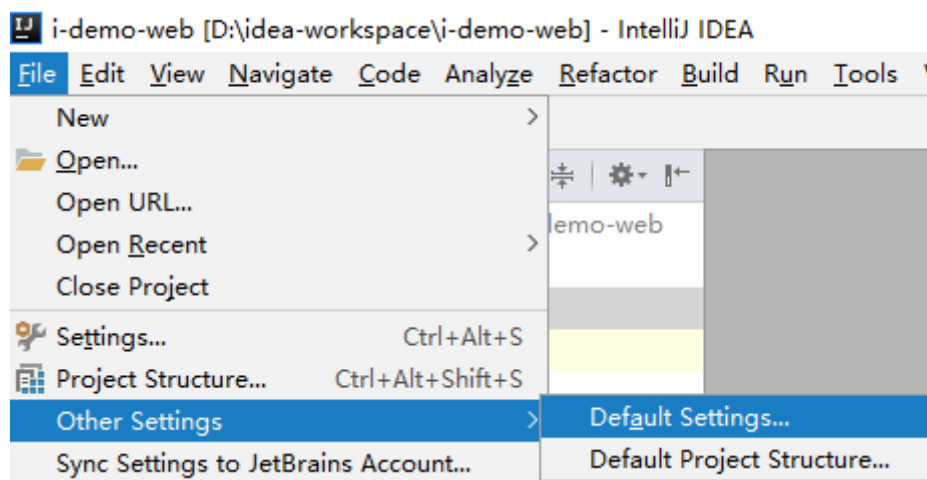


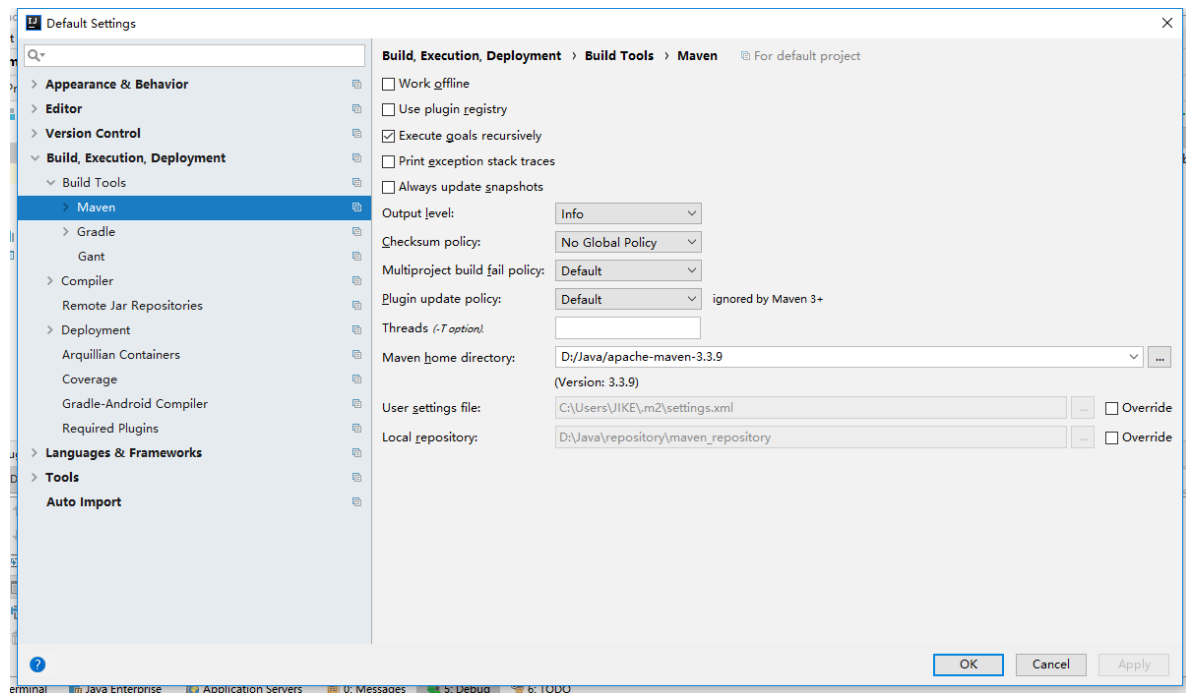
Packaging: 指定打包方式，默认为jar。选项有：jar、war、pom。

第四步：点击finish，完成maven工程创建。

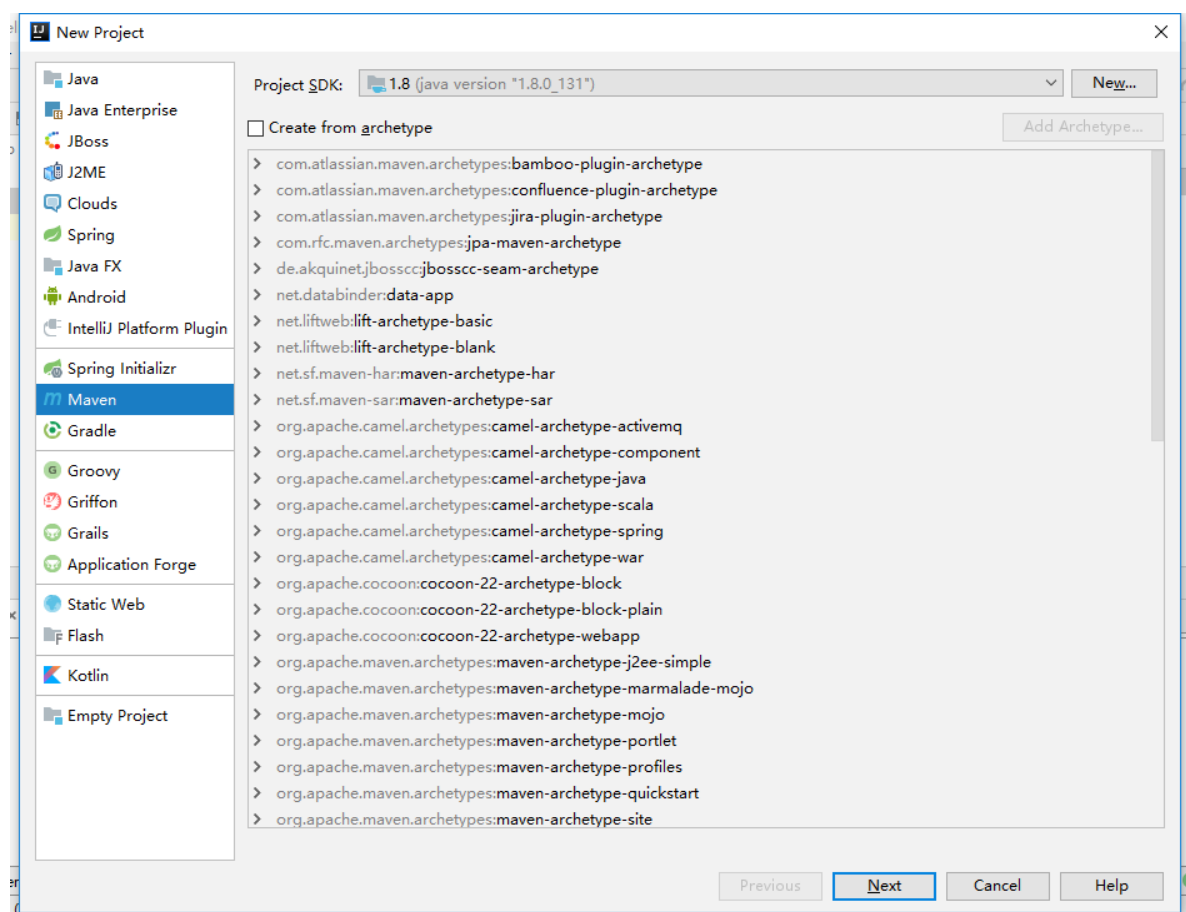
5. IDEA集成

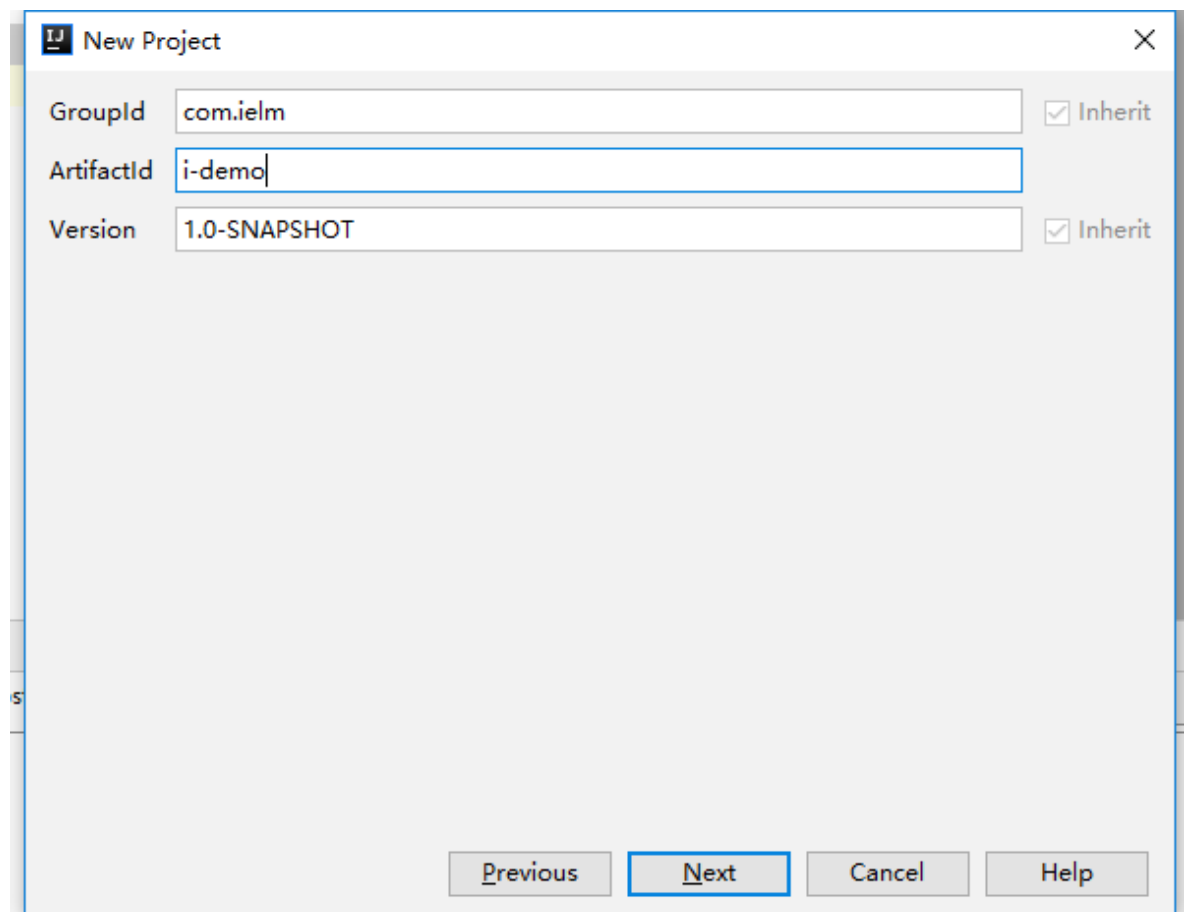
5.1 设置maven安装目录





5.2 创建maven工程





The 'New Project' dialog box shows the first step of project creation. It includes input fields for 'GroupId' (com.ielm), 'ArtifactId' (i-demo), and 'Version' (1.0-SNAPSHOT). There are checkboxes for 'Inherit' next to the 'GroupId' and 'Version' fields. At the bottom, there are buttons for 'Previous', 'Next', 'Cancel', and 'Help'.

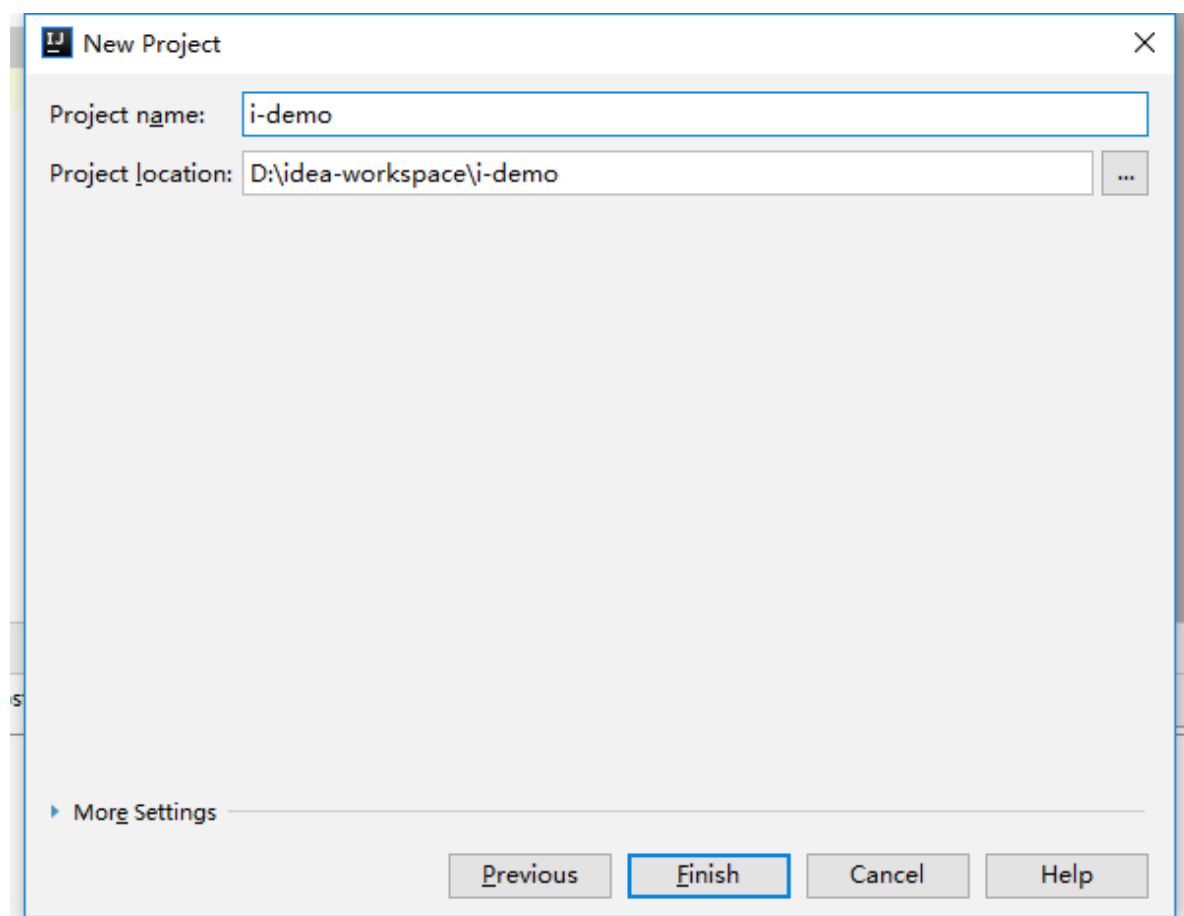
New Project

GroupId: com.ielm ☒ Inherit

ArtifactId: i-demo

Version: 1.0-SNAPSHOT ☒ Inherit

Previous Next Cancel Help



The 'New Project' dialog box shows the second step of project creation. It includes input fields for 'Project name' (i-demo) and 'Project location' (D:\idea-workspace\i-demo). There is a button with three dots next to the 'Project location' field. At the bottom, there is a 'More Settings' link and buttons for 'Previous', 'Finish', 'Cancel', and 'Help'.

New Project

Project name: i-demo

Project location: D:\idea-workspace\i-demo ...

More Settings

Previous Finish Cancel Help

6. Maven 命令

需要在pom.xml所在目录中执行以下命令。

6.1 mvn compile

执行 mvn compile命令，完成编译操作

执行完毕后，会生成target目录，该目录中存放了编译后的字节码文件。

6.2 mvn clean

执行 mvn clean命令

执行完毕后，会将target目录删除。

6.3 mvn test

执行 mvn test命令，完成单元测试操作

执行完毕后，会在target目录中生成三个文件夹：surefire、surefire-reports（测试报告）、test-classes（测试的字节码文件）

6.4 mvn package

执行 mvn package命令，完成打包操作

执行完毕后，会在target目录中生成一个文件，该文件可能是jar、war

6.5 mvn install

执行 mvn install命令，完成将打好的jar包安装到本地仓库的操作

执行完毕后，会在本地仓库中出现安装后的jar包，方便其他工程引用

6.6 mvn clean compile命令

cmd 中录入 mvn clean compile命令

组合指令，先执行clean，再执行compile，通常应用于上线前执行，清除测试类

6.7 mvn clean test命令

cmd 中录入 mvn clean test命令

组合指令，先执行clean，再执行test，通常应用于测试环节

6.8 mvn clean package命令

cmd 中录入 mvn clean package命令

组合指令，先执行clean，再执行package，将项目打包，通常应用于发布前

执行过程：

清理———清空环境

编译———编译源码

测试———测试源码

打包———将编译的非测试类打包

6.9 mvn clean install命令

cmd 中录入 mvn clean install 查看仓库，当前项目被发布到本地仓库中

组合指令，先执行clean，再执行install，将项目打包

执行过程：

清理———清空环境

编译———编译源码

测试———测试源码

打包———将编译的非测试类打包

安装———将打好的包发布到本地仓库中

6.10 跳过测试

```
1 -Dmaven.test.skip=true
2 mvn package -Dmaven.test.skip=true
```

7. Maven 相关概念

7.1 坐标

什么是坐标？

在平面几何中坐标 (x,y) 可以标识平面中唯一的一点。在maven中坐标就是为了定位一个唯一确定的jar包。

Maven世界拥有大量构建，我们需要找一个用来唯一标识一个构建的统一规范

拥有了统一规范，就可以把查找工作交给机器

Maven坐标

groupId：定义当前Maven组织名称

artifactId：定义实际项目名称

version：定义当前项目的当前版本

7.2 依赖管理

就是对项目中jar包的管理。可以在pom文件中定义jar包的GAV坐标，管理依赖。

依赖声明主要包含如下元素：

```
1 <dependency>
2   <groupId>junit</groupId>
3   <artifactId>junit</artifactId>
4   <version>4.12</version>
5   <scope>test</scope>
6 </dependency>
```

依赖范围

依赖范围 (Scope)	对于主代码 classpath有效	对于测试代码 classpath有效	被打包，对于 运行时 classpath有效	例子
compile	Y	Y	Y	log4j
test	-	Y	-	junit
provided	Y	Y	-	servlet-api
runtime	-	-	Y	JDBC Driver Implementation

其中依赖范围scope 用来控制依赖和编译，测试，运行的classpath的关系. 主要的是三种依赖关系如下：

- compile**：默认编译依赖范围。对于编译，测试，运行三种classpath都有效
- test**：测试依赖范围。只对于测试classpath有效
- provided**：已提供依赖范围。对于编译，测试的classpath都有效，但对于运行无效。因为由容器已经提供，例如servlet-api
- runtime**：运行时提供。例如:jdbc驱动

依赖传递



如果B中使用A，C中使用B，则称B是C的**直接依赖**，而称A是C的**间接依赖**。

C->B B->A

C直接依赖B

C间接依赖A

依赖冲突

- 如果直接依赖与间接依赖中包含有同一个坐标不同版本的资源依赖，以直接依赖的版本为准（就近原则）
- 如果直接依赖中包含有同一个坐标不同版本的资源依赖，以配置顺序下方的版本为准（就近原则）

可选依赖

```
1 <optional> true/false </optional>
```

是否可选，也可以理解为是否向下传递。

在依赖中添加optional选项决定此依赖是否向下传递，如果是true则不传递，如果是false就传递，默认为false。

```
1 <dependency>
2   <groupId>junit</groupId>
3   <artifactId>junit</artifactId>
4   <version>4.12</version>
5   <scope>test</scope>
6
7   <!--
8       可选依赖
9       在依赖中添加optional选项决定此依赖是否向下传递，
10      如果是true则不传递，如果是false就传递，默认为false
11   -->
12   <optional>false</optional>
13 </dependency>
```

排除依赖

排除依赖包中所包含的依赖关系，**不需要添加版本号**。

如果在本次依赖中有一些多余的jar包也被传递依赖过来，如果想把这些jar包排除的话可以配置exclusions进行排除。

```
1 <dependency>
2   <groupId>junit</groupId>
3   <artifactId>junit</artifactId>
4   <version>4.12</version>
5   <scope>test</scope>
6
7   <!--
8       可选依赖
9       在依赖中添加optional选项决定此依赖是否向下传递，
10      如果是true则不传递，如果是false就传递，默认为false
11   -->
12   <optional>false</optional>
13
14   <!--排除依赖-->
15   <exclusions>
16     <exclusion>
17       <groupId>org.hamcrest</groupId>
```

```
18         <artifactId>hamcrest-core</artifactId>
19         </exclusion>
20     </exclusions>
21 </dependency>
```

7.3 生命周期

什么是生命周期？

Maven生命周期就是为了**对所有的构建过程进行抽象和统一**。包括项目清理、初始化、编译、打包、测试、部署等几乎所有构建步骤。

生命周期可以理解为构建工程的步骤。

在Maven中有三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，这三套生命周期分别是：

- **Clean Lifecycle**：在进行真正的构建之前进行一些清理工作。
- **Default Lifecycle**：构建的核心部分，**编译，测试，打包，部署**等等。
- **Site Lifecycle**：生成项目报告，站点，发布站点。

再次强调一下它们是相互独立的，你可以仅仅调用clean来清理工作目录，仅仅调用site来生成站点。当然你也可以直接运行 `mvn clean install site` 运行所有这三套生命周期。

Maven三大生命周期

clean：清理项目

每套生命周期都由一组阶段(Phase)组成，我们平时在命令行输入的命令总会对应于一个特定的阶段。比如，运行`mvn clean`，这个的clean是Clean生命周期的一个阶段。有Clean生命周期，也有clean阶段。Clean生命周期一共包含了三个阶段：

pre-clean 执行一些需要在clean之前完成的工作

clean 移除所有上一次构建生成的文件

post-clean 执行一些需要在clean之后立刻完成的工作

`mvn clean` 中的clean就是上面的clean，在一个生命周期中，运行某个阶段的时候，它之前的所有阶段都会被运行，也就是说，`mvn clean` 等同于 `mvn pre-clean clean`，如果我们运行 `mvn post-clean`，那么 `pre-clean`，`clean` 都会被运行。这是Maven很重要的一个规则，可以大大简化命令行的输入。

default：构建项目

Default生命周期是Maven生命周期中最重要的一个，绝大部分工作都发生在这个生命周期中。这里，只解释一些比较重要和常用的阶段：

validate

generate-sources

process-sources

generate-resources

process-resources 复制并处理资源文件，至目标目录。

compile 编译项目的源代码。

process-classes

generate-test-sources

process-test-sources

generate-test-resources

process-test-resources 复制并处理字节码文件，至目标目录。

test-compile 编译测试源代码。

process-test-classes

test 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。

prepare-package

package 接受编译好的代码，打包成可发布的格式，如 jar、war。

pre-integration-test

integration-test

post-integration-test

verify

install 将包安装至本地仓库，以让其它项目依赖。

deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

运行任何一个阶段的时候，它前面的所有阶段都会被运行，这也就是为什么我们运行 `mvn install` 的时候，代码会被编译，测试，打包。

site：生成项目站点

site生命周期

pre-site 执行一些需要在生成站点文档之前完成的工作

site 生成项目的站点文档

post-site 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备

site-deploy 将生成的站点文档部署到特定的服务器上

这里经常用到的是site阶段和site-deploy阶段，用以生成和发布Maven站点，这可是Maven相当强大的功能。

7.3 Maven插件

Maven的核心仅仅定义了抽象的生命周期，具体的任务都是交由插件完成的。每个插件都能实现一个功能，每个功能就是一个插件目标。Maven的生命周期与插件目标相互绑定，以完成某个具体的构建任务。

例如compile就是插件maven-compiler-plugin的一个插件目标

```
1     <build>
2         <plugins>
3             <plugin>
4                 <groupId>org.apache.maven.plugins</groupId>
```

```

5         <artifactId>maven-resources-plugin</artifactId>
6         <version>3.1.0</version>
7     </plugin>
8     <plugin>
9         <groupId>org.apache.maven.plugins</groupId>
10        <artifactId>maven-compiler-plugin</artifactId>
11        <version>3.8.0</version>
12    </plugin>
13
14    <plugin>
15        <groupId>org.apache.maven.plugins</groupId>
16        <artifactId>maven-install-plugin</artifactId>
17        <version>2.5.2</version>
18    </plugin>
19    <plugin>
20        <groupId>org.apache.maven.plugins</groupId>
21        <artifactId>maven-surefire-plugin</artifactId>
22        <version>2.19.1</version>
23        <configuration>
24            <!--跳过测试用例的运行-->
25            <skip>true</skip>
26        </configuration>
27    </plugin>
28 </plugins>
29 </build>

```


7.4 继承

继承是为了消除重复，可以把很多相同的配置提取出来。例如：groupId，version等

创建父工程

新建项目的过程和创建普通maven工程一样，

创建成功以后在pom.xml中设置packaging标签的内容为pom

 i-maven-parent x

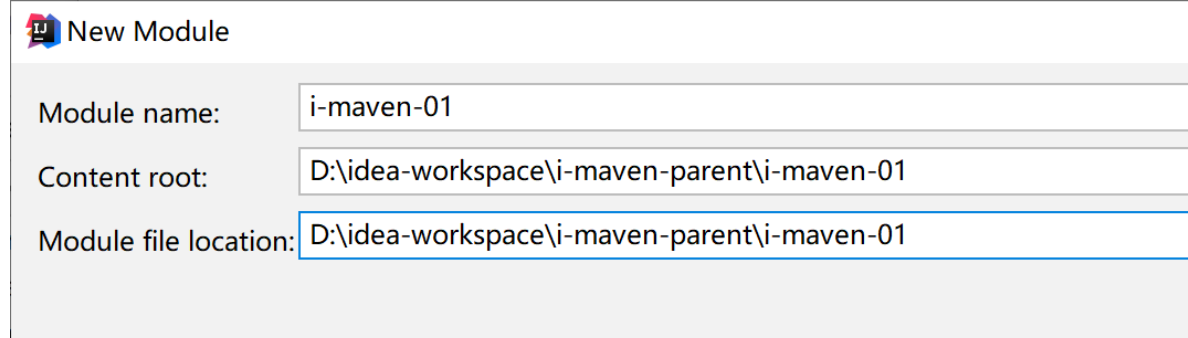
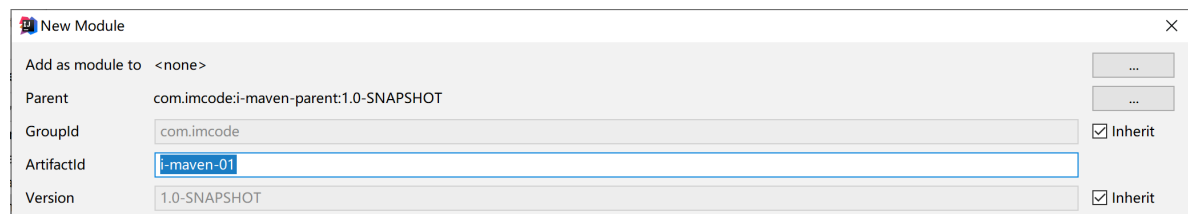
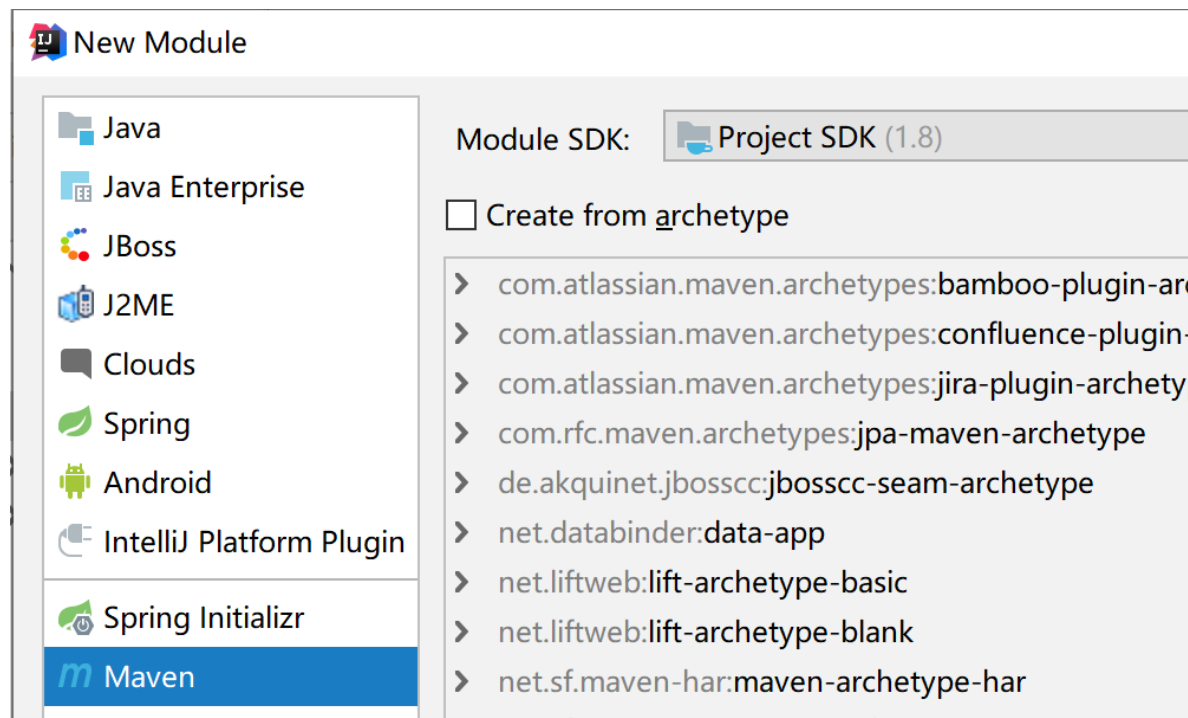
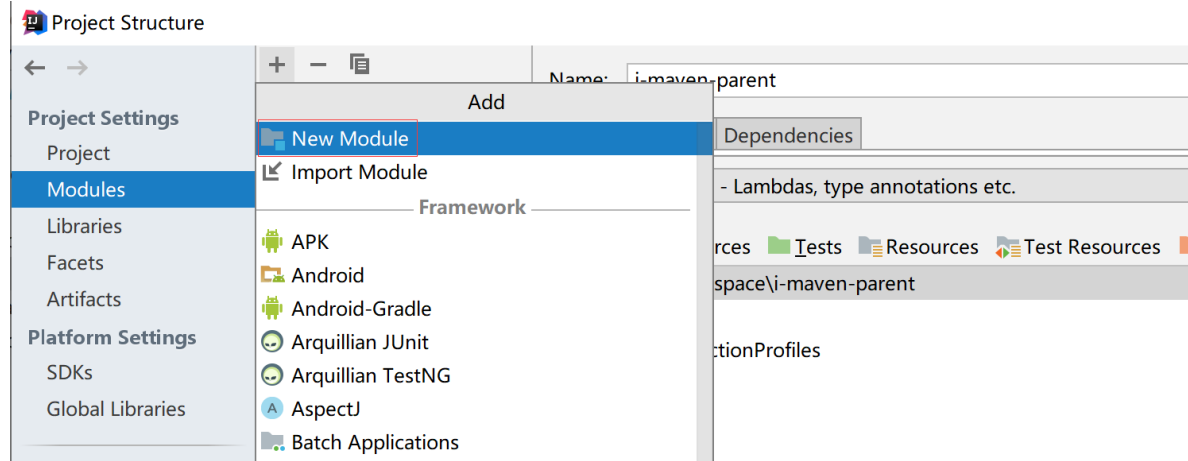
```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>com.imcode</groupId>
8      <artifactId>i-maven-parent</artifactId>
9      <version>1.0-SNAPSHOT</version>
10     <packaging>pom</packaging>
11
12
13 </project>

```

创建子工程

现有工程继承父工程只需要在pom文件中添加parent节点即可。



父工程统一管理版本号

Maven使用dependencyManagement管理依赖的版本号。

注意：此处只是定义依赖jar包的版本号，并不实际依赖。如果子工程中需要依赖jar包还需要添加dependency节点。

父工程pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5          http://maven.apache.org/xsd/maven-4.0.0.xsd">
6
7      <groupId>com.imcode</groupId>
8      <version>1.0-SNAPSHOT</version>
9      <artifactId>i-maven-parent</artifactId>
10     <packaging>pom</packaging>
11     <properties>
12         <junit.version>4.12</junit.version>
13     </properties>
14     <!-- 依赖声明标签-->
15     <dependencyManagement>
16         <dependencies>
17             <dependency>
18                 <groupId>junit</groupId>
19                 <artifactId>junit</artifactId>
20                 <version>${junit.version}</version>
21                 <scope>test</scope>
22             </dependency>
23         </dependencies>
24     </dependencyManagement>
25 </project>
```

子工程pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5          http://maven.apache.org/xsd/maven-4.0.0.xsd">
6      <modelVersion>4.0.0</modelVersion>
7      <parent>
8          <artifactId>i-maven-parent</artifactId>
9          <groupId>com.imcode</groupId>
10         <version>1.0-SNAPSHOT</version>
11     </parent>
12     <artifactId>i-maven-01</artifactId>
13
14     <dependencies>
15         <dependency>
16             <groupId>junit</groupId>
17             <artifactId>junit</artifactId>
18         </dependency>
19     </dependencies>
20 </project>
```


7.5 聚合

聚合一般是一个工程拆分成多个模块开发，每个模块是一个独立的工程，但是要是运行时必须把所有模块聚合到一起才是一个完整的工程，此时可以使用maven的聚合工程。

例如电商项目中，包括商品模块、订单模块、用户模块等。就可以对不同的模块单独创建工程，最终在打包时，将不同的模块聚合到一起。（按业务模块聚合）

例如同一个项目中的表现层、业务层、持久层，也可以分层创建不同的工程，最后打包运行时，再聚合到一起。（架构上的聚合）

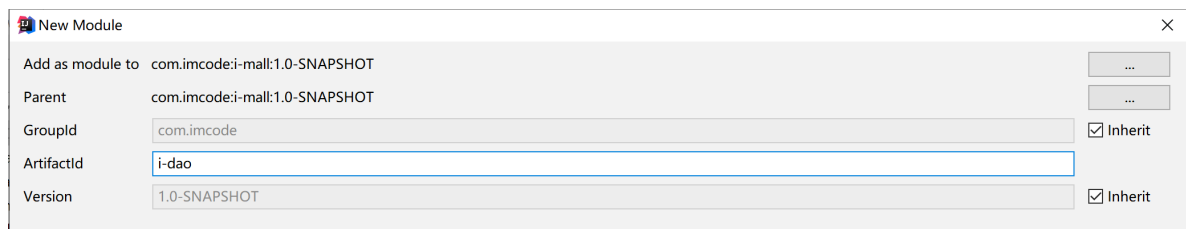
```
1    i-mall
2
3        i-dao 持久层    独立的工程
4
5        i-goods 商品模块 依赖持久层    业务层 + 表现层    独立的工程
6
7        i-order 订单模块 依赖持久层    业务层 + 表现层    独立的工程
8        ....
```

创建一个聚合工程

聚合工程的打包方式必须是pom，一般聚合工程和父工程合并为一个工程。

第一步：创建父工程 i-mall

第二步：创建 i-dao 持久层工程



第三步：检查聚合工程(父工程)的pom.xml



8. Maven仓库(了解)

8.1 什么是Maven仓库?

用来统一存储所有Maven共享构建的位置就是仓库。根据Maven坐标定义每个构建在仓库中唯一存储路径大致为：groupId/artifactId/version/artifactId-version.packaging

8.2 仓库的分类

1、本地仓库

~/m2/repository

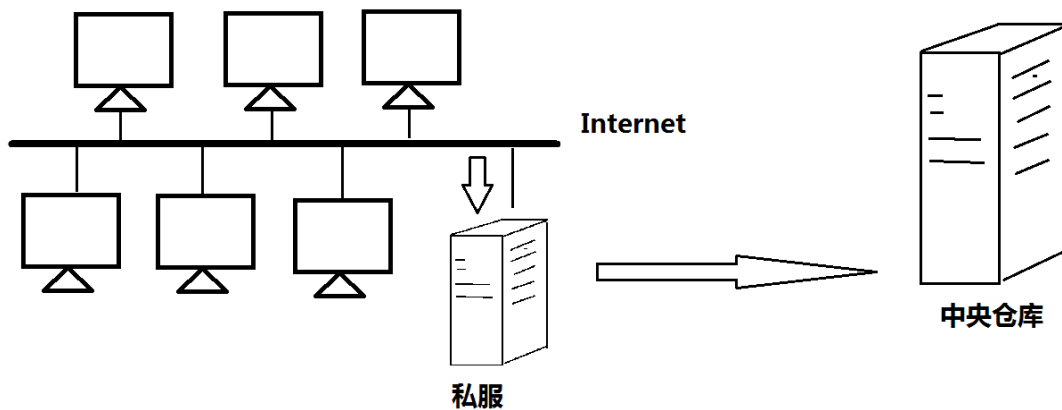
每个用户只有一个本地仓库

2、远程仓库

- 中央仓库：Maven默认的远程仓库，不包含版权资源

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

- 私服：是一种特殊的远程仓库，它是架设在局域网内的仓库



8.3 Maven私服

安装Nexus

为所有来自中央仓库的构建安装提供本地缓存。

下载网站：<http://nexus.sonatype.org/>

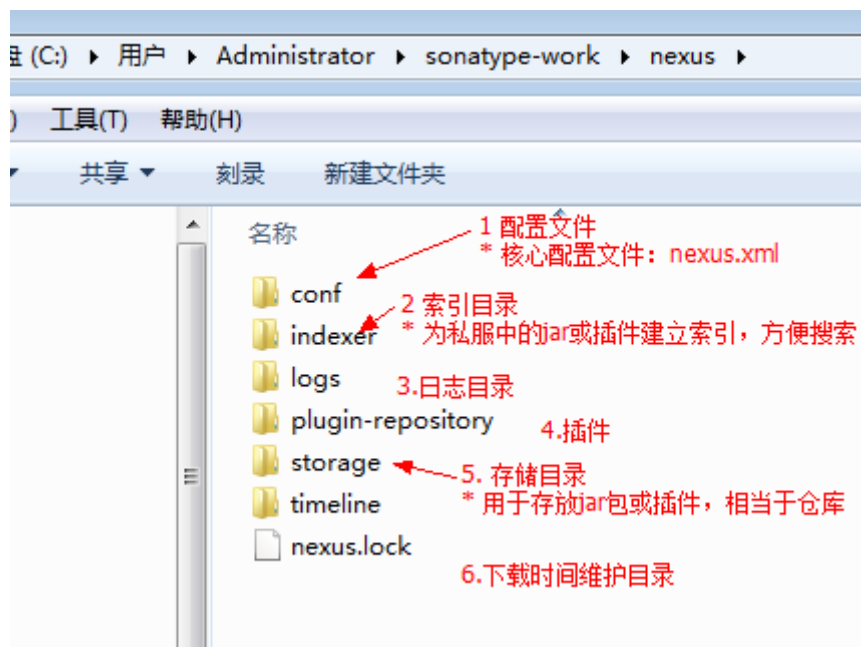
安装版本：nexus-2.7.0-06.war

第一步：将下载的nexus的war包复制到tomcat下的webapps目录。

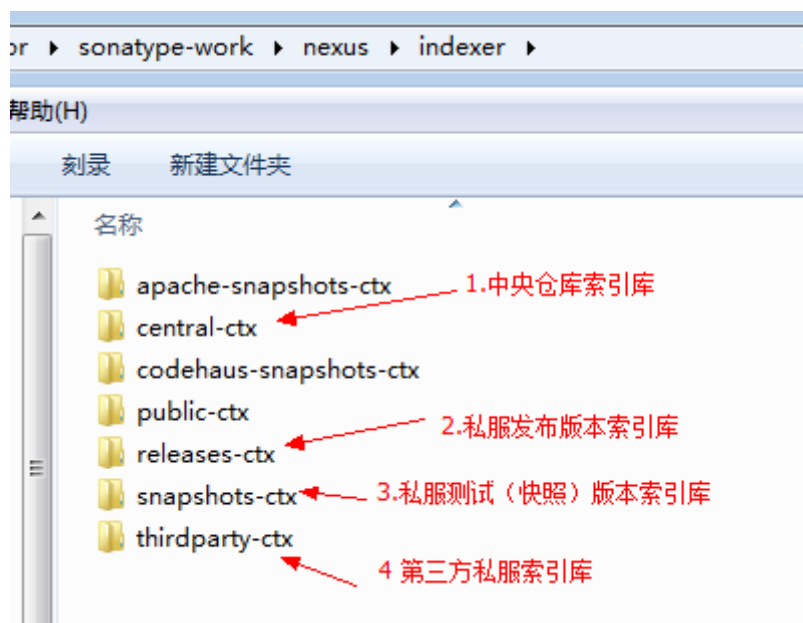
第二步：启动tomcat。nexus将在c盘创建sonatype-work目录【C:\Users\当前用户\sonatype-work\nexus】。

Nexus的目录结构

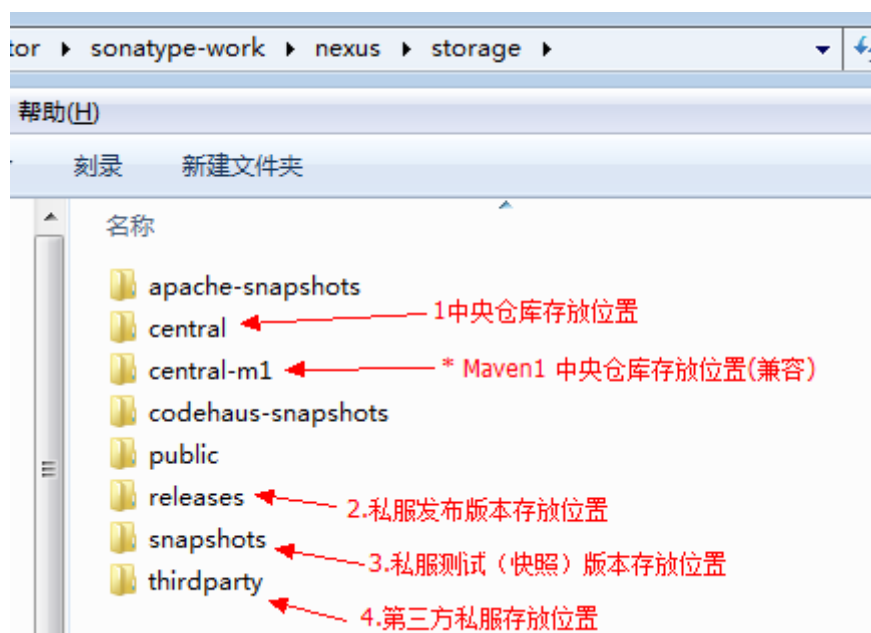
- 目录结构如下：



- Indexer 索引目录结构:



- Storage 存储目录结构:



访问Nexus

访问URL: <http://localhost:8080/nexus-2.7.0-06/>

默认账号:

用户名: admin

密码: admin123

Nexus的仓库和仓库组

Welcome

Repositories

Search

Refresh

Add...

Delete

Trash...

User Managed Repositories

Repository	Type	Quality	Format	Policy	Repository Status
Public Repositories	group	ANALYZE	maven2		
3rd party	hosted	ANALYZE	maven2	Release	In Service
Apache Snapshots	proxy	ANALYZE	maven2	Snapshot	In Service
Central	proxy	ANALYZE	maven2	Release	In Service
Central M1 shadow	virtual	ANALYZE	maven1	Release	In Service
Codehaus Snapshots	proxy	ANALYZE	maven2	Snapshot	In Service
Releases	hosted	ANALYZE	maven2	Release	In Service
Snapshots	hosted	ANALYZE	maven2	Snapshot	In Service

仓库有4种类型:

- group(仓库组): 一组仓库的集合
- hosted(宿主): 配置第三方仓库 (包括公司内部私服)
- proxy(代理): 私服会对中央仓库进行代理, 用户连接私服, 私服自动去中央仓库下载jar包或者插件
- virtual(虚拟): 兼容Maven1 版本的jar或者插件

Nexus的仓库和仓库组介绍:

- 3rd party: 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建
- Apache Snapshots: 一个策略为Snapshot的代理仓库, 用来代理Apache Maven仓库的快照版本构建
- Central: 代理Maven中央仓库
- Central M1 shadow: 代理Maven1 版本 中央仓库
- Codehaus Snapshots: 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
- Releases: 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: 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
- 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

配置所有构建均从私服下载

在本地仓库的setting.xml中配置如下:

```
| <mirrors> <mirror> <!--此处配置所有的构建均从私有仓库中下载 *代表所有, 也可以写central -->
<id>nexus</id> <mirrorOf>*</mirrorOf> <url> http://localhost:8080/nexus-2.7.0-06/content/groups/public/ </url> </mirror> </mirrors> | |-----
|
```

```

<!-- mirror
| Specifies a repository mirror site to use instead of a given repository. Th
| this mirror serves has an ID that matches the mirrorOf element of this mirr
| for inheritance and direct lookup purposes, and must be unique across the s
|-->
<mirror>
  <id>nexus</id>
  <mirrorOf>*</mirrorOf>
  <name>nexus</name>
  <url>http://localhost:8080/nexus-2.7.0-06/content/groups/public/</url>
</mirror>
</mirrors>

```

可以配置*或者central

部署构建到Nexus

第一步：Nexus的访问权限控制

在本地仓库的setting.xml中配置如下：

```

<server> <id>releases</id> <username>admin</username> <password>admin123</password> </server>
<server> <id>snapshots</id> <username>admin</username> <password>admin123</password> </server>
-----

```

第二步：配置pom文件

在需要构建的项目中修改pom文件

```

<distributionManagement> <repository> <id>releases</id> <name>Internal Releases</name> <url> http://l
ocalhost:8080/nexus-2.7.0-06/content/repositories/releases/ </url> </repository> <snapshotRepository>
<id>snapshots</id> <name>Internal Snapshots</name> <url> http://localhost:8080/nexus-2.7.0-06/conten
t/repositories/snapshots/ </url> </snapshotRepository> </distributionManagement> | -----
-----
-----

```

第三步：执行maven的deploy命令

