Maven-项目管理工具

课程计划

1、maven介绍
2、maven配置
3、创建maven工程
4、maven的核心概念

1、maven介绍

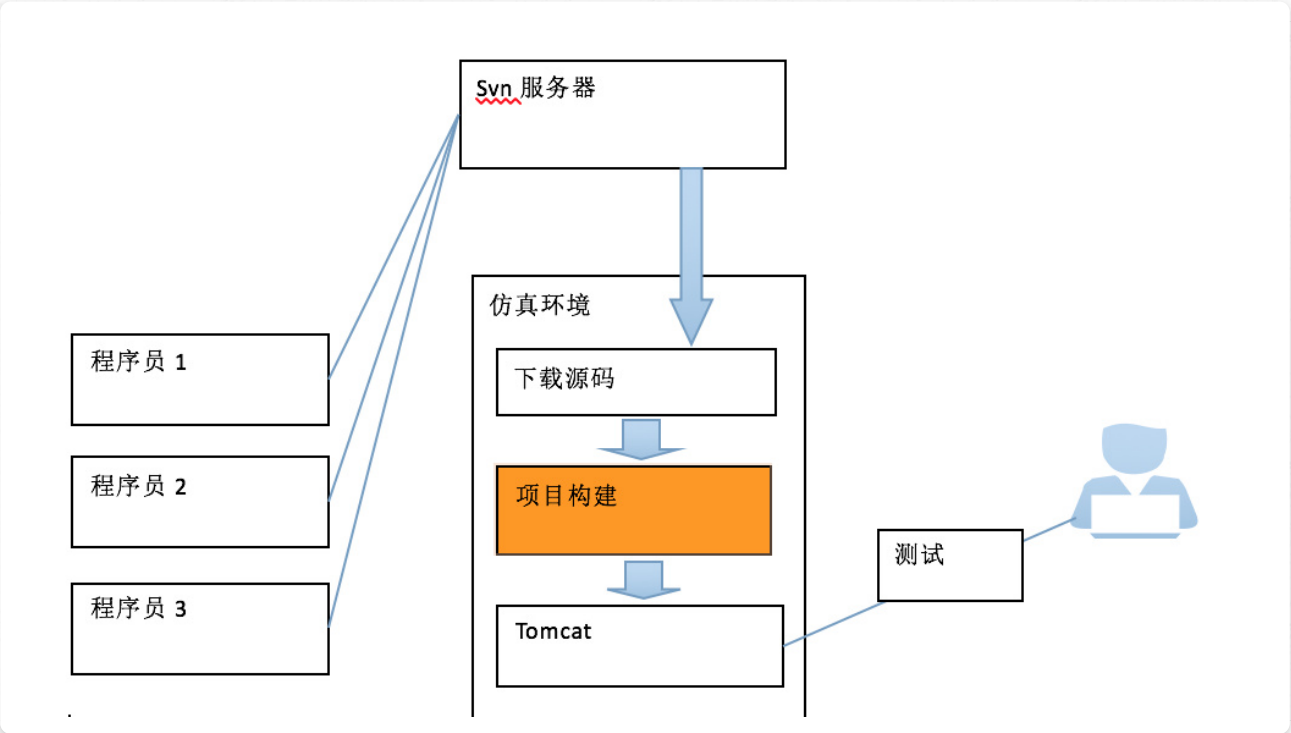
1.1 开发中遇到的问题

- 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?
- 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?
- 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。
- 我的项目依赖一些jar包，我应该把他们放哪里？放源码库里?
- 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
- 现在是第三次，再复制一次吧 ----- 这样真的好吗?
- 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
- 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。

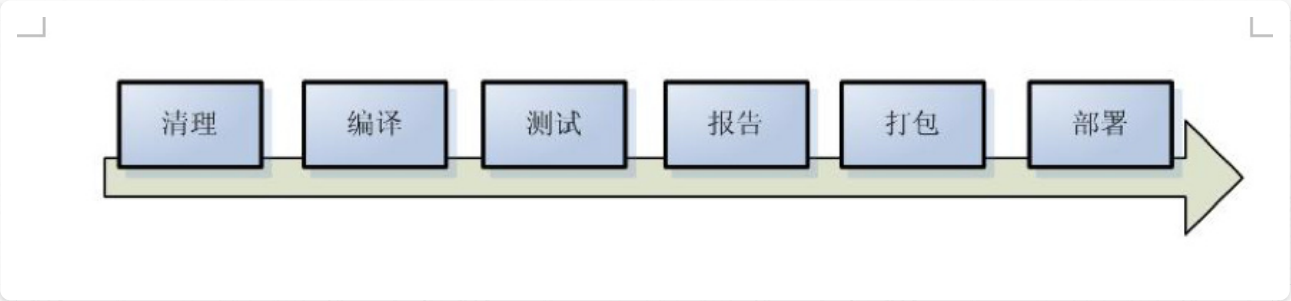
1.2 什么是maven

- Maven是基于POM（工程对象模型），通过一小段描述来对项目的代码、报告、文件进行管理的工具。
- Maven是一个跨平台的项目管理工具，它是使用java开发的，它要依赖于jdk1.6及以上
- Maven主要有两大功能：管理依赖、项目构建。
- 依赖指的就是jar包。

1.3 什么是构建



- 构建过程：



2、maven的配置

2-1.下载并解压

- 2-1-1.下载地址：maven下载地址；选择`Binary tar.gz archive`或者`Binary zip archive`都可以。
- 解压到任意地址，例如：`/Users/lizhongren1/maven/apache-maven-3.3.9`，下面统一使用该路径配置maven。各位读者请各自选择地址。

2-2.配置环境变量

2-2-1.maven的环境变量

- 终端输入`$ open .bash_profile`
- 将下面的三行环境变量输入，保存并关闭（确保Java环境变量已经配置）：

```
export M2_HOME=/Users/lizhongren1/maven/apache-maven-3.3.9
export M2=$M2_HOME/bin
```

```
export PATH=$M2:$PATH
```

- 终端输入`$ source .bash_profile`,使修改生效。

2-2-2. 检查是否安装成功

终端输入`mvn -v`.显示maven的配置信息则成功。

2-3.修改maven配置

2-3-1.配置本地库

- 在`/Users/lizhongren1/maven/`路径下新建一个`LocalWarehouse`文件夹作为本地仓库地址
- 打开`/Users/lizhongren1/maven/apache-maven-3.3.9/conf/settings.xml`,在setting标签中加入一行:

```
<localRepository>/Users/lizhongren1/maven/LocalWarehouse</localReposito
ry>
```

2-3-2.配置镜像

- 上面打开的setting.xml文件中，找到下面mirror标签加入阿里的镜像信息:

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

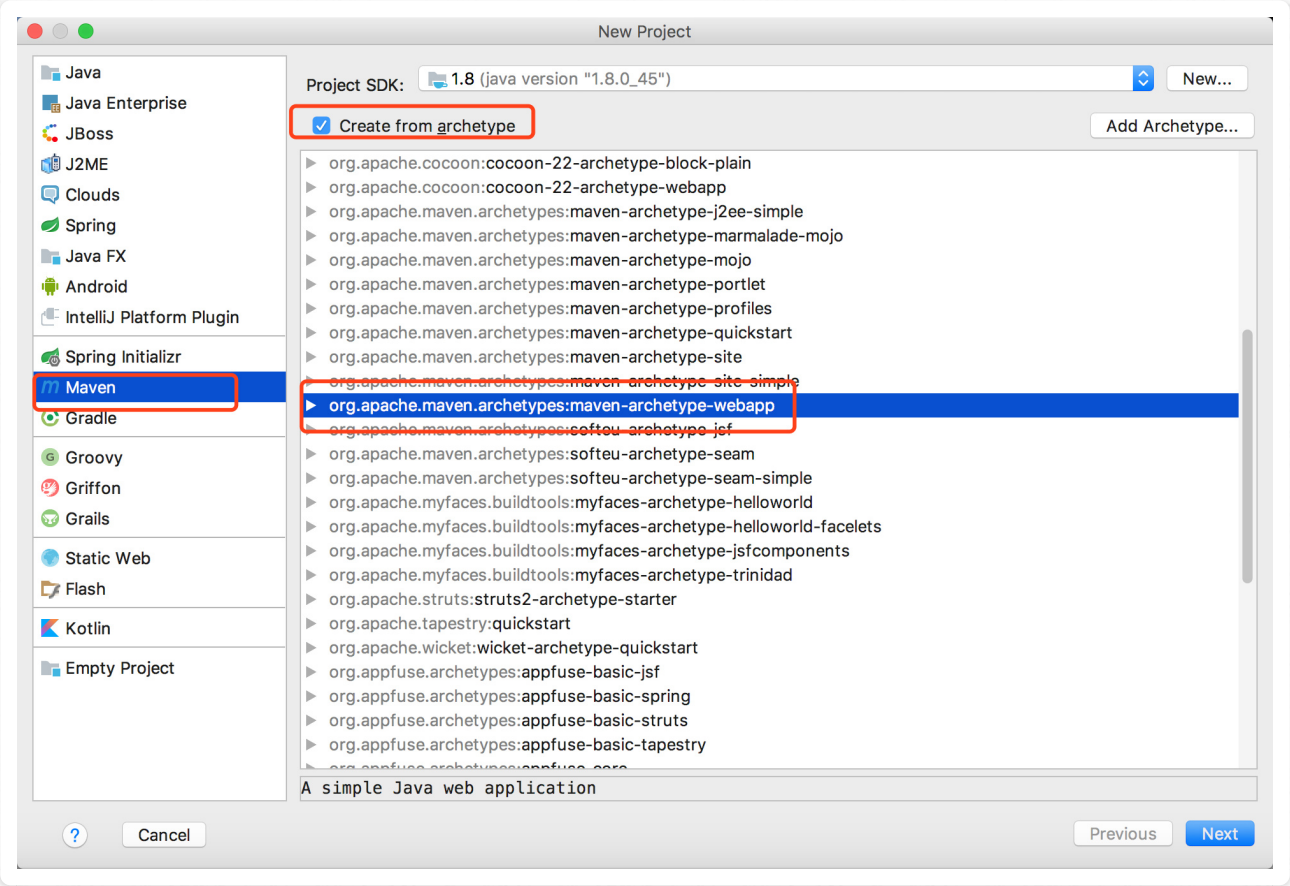
2-4.配置idea的maven

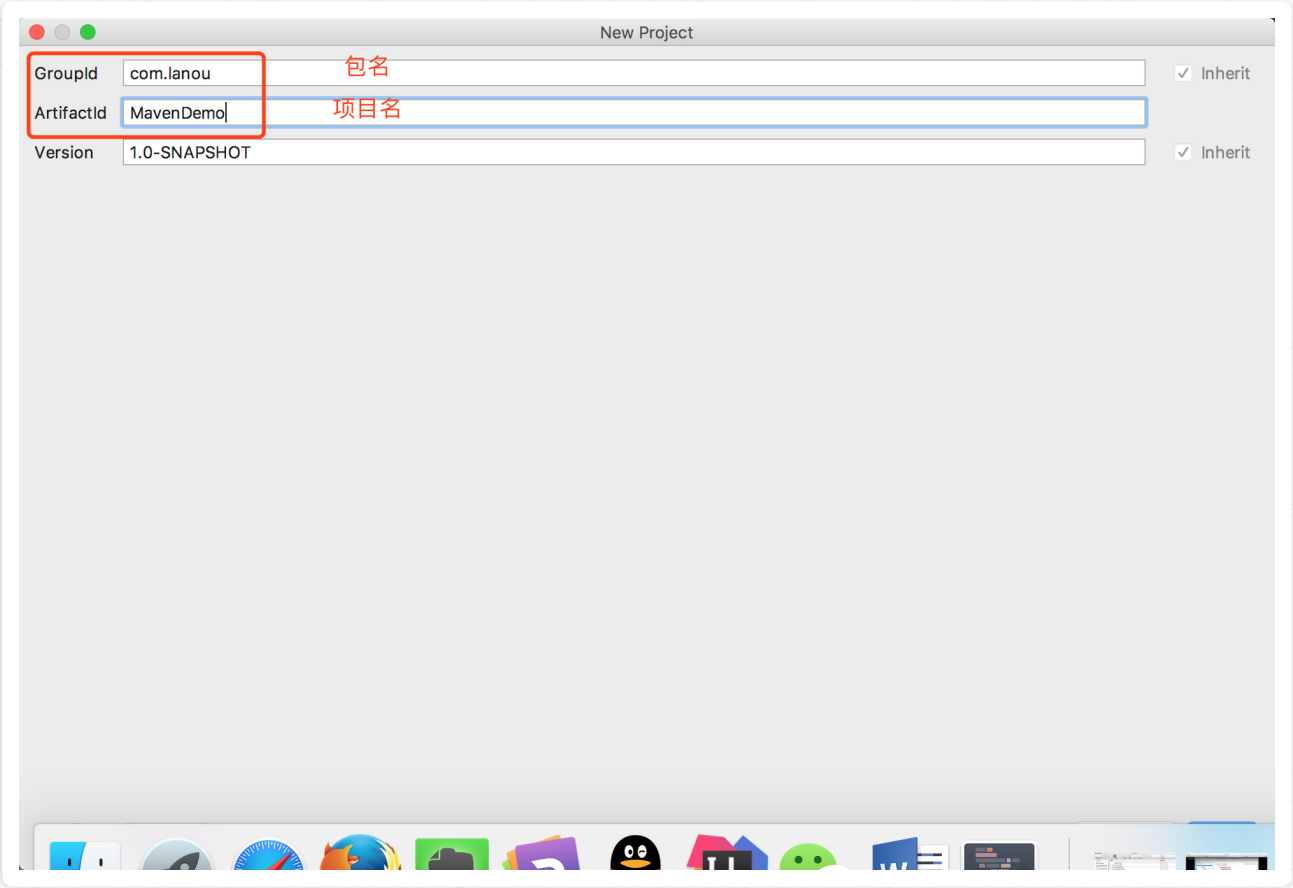
- 打开`preference`，找到`Build-Build Tools-Maven`。这里有自带的maven框架
- 将`Maven home directory`的地址改为`/Users/lizhongren1/maven/apache-maven-3.3.9`
- 将`User setting file`修改为`/Users/lizhongren1/maven/apache-maven-3.3.9/conf/settings.xml`
- 保存即可

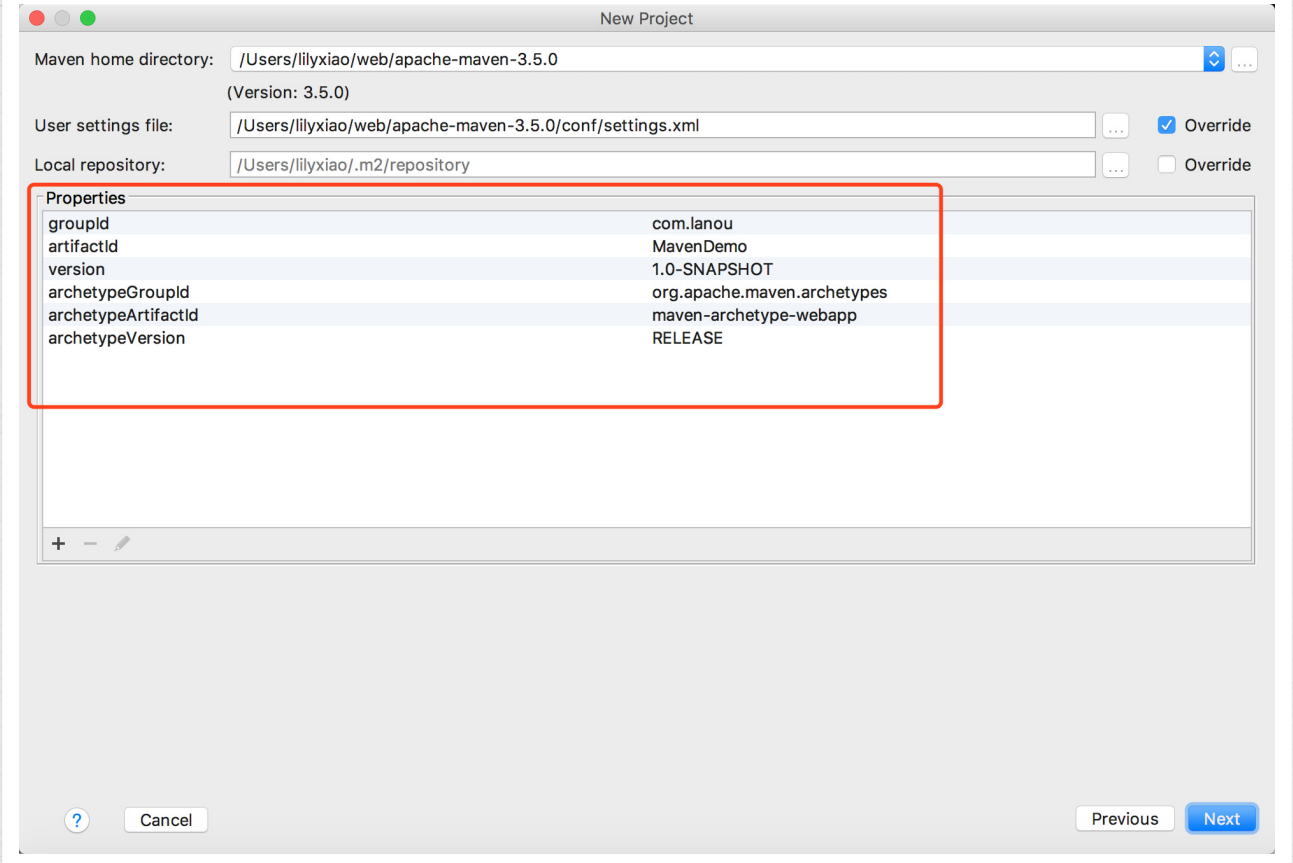
3、创建maven工程

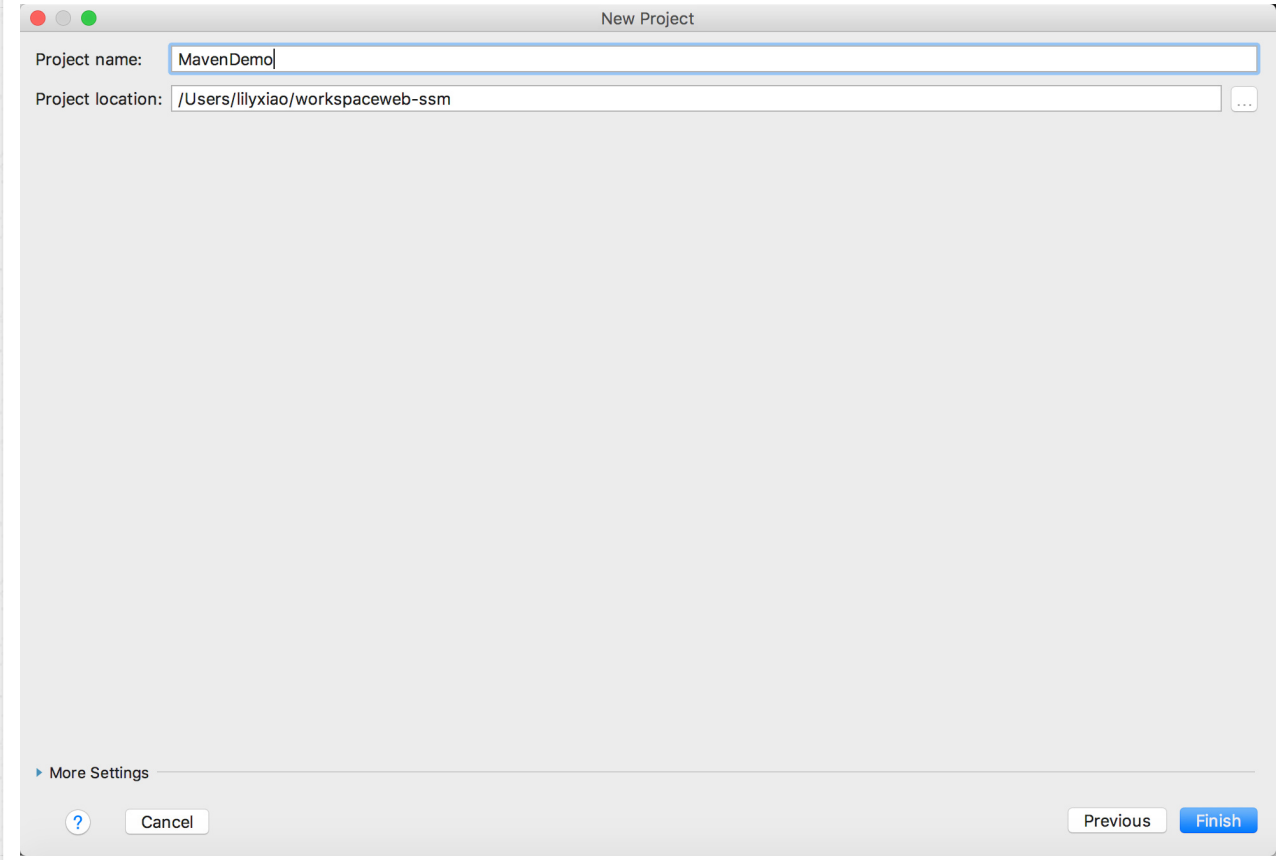
3-1. 创建项目

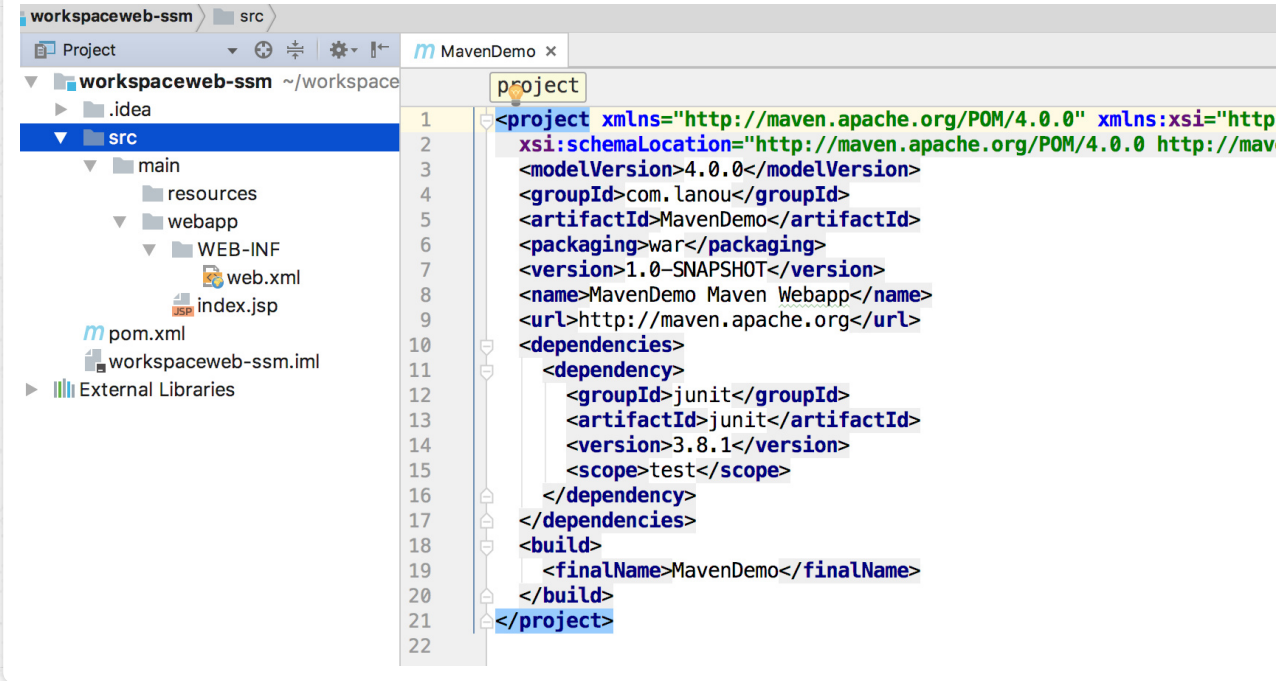
- `mvn archetype:create -DgroupId=packageName -DartifactId=projectName`
- `mvn archetype:create -DgroupId=packageName -DartifactId=webappName -DarchetypeArtifactId=maven-archetype-webapp`

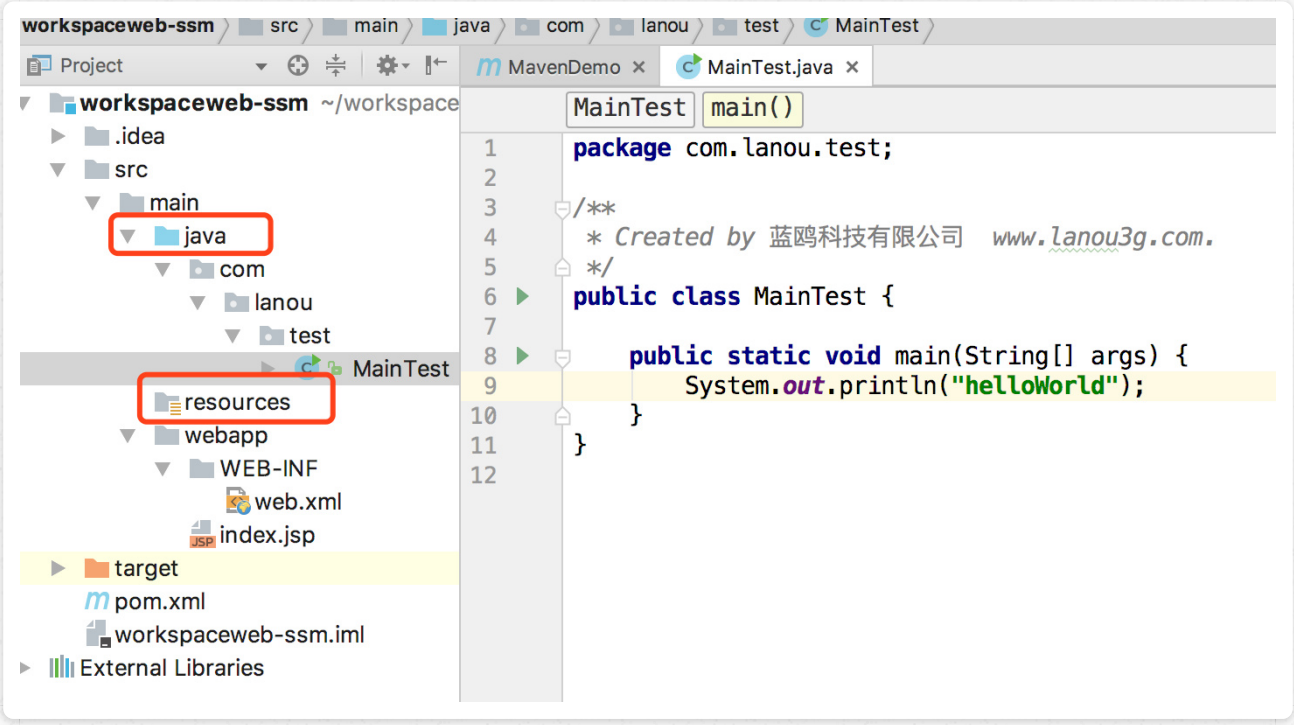



New Project
GroupId
com.lanou
包名
Inherit
ArtifactId
MavenDemo
项目名
Version
1.0-SNAPSHOT
Inherit


New Project
Maven home directory:
/Users/lilyxiao/web/apache-maven-3.5.0
(Version: 3.5.0)
User settings file:
/Users/lilyxiao/web/apache-maven-3.5.0/conf/settings.xml
Override
Local repository:
/Users/lilyxiao/.m2/repository
Override
Properties
groupId com.lanou
artifactId MavenDemo
version 1.0-SNAPSHOT
archetypeGroupId org.apache.maven.archetypes
archetypeArtifactId maven-archetype-webapp
archetypeVersion RELEASE
Cancel
Previous
Next


New Project
Project name:
MavenDemo
Project location:
/Users/lilyxiao/workspaceweb-ssm
More Settings
Cancel
Previous
Finish


workspaceweb-ssm
src
Project
MavenDemo
workspaceweb-ssm ~/workspace
.idea
src
main
resources
webapp
WEB-INF
web.xml
index.jsp
pom.xml
workspaceweb-ssm.iml
External Libraries
project
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http
2 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://mav
3 <modelVersion>4.0.0</modelVersion>
4 <groupId>com.lanou</groupId>
5 <artifactId>MavenDemo</artifactId>
6 <packaging>war</packaging>
7 <version>1.0-SNAPSHOT</version>
8 <name>MavenDemo Maven Webapp</name>
9 <url>http://maven.apache.org</url>
10 <dependencies>
11 <dependency>
12 <groupId>junit</groupId>
13 <artifactId>junit</artifactId>
14 <version>3.8.1</version>
15 <scope>test</scope>
16 </dependency>
17 </dependencies>
18 <build>
19 <finalName>MavenDemo</finalName>
20 </build>
21 </project>
22




```
├──src
│   ├──main
│   │   └──java
│   │       └──cnblogs
│   └──test
│       └──java
│           └──cnblogs
└──target
    └──classes
        └──cnblogs
```

注意上面带红色的目录名，maven项目采用"约定优于配置"的原则，src/main/java约定用于存放源代码，src/main/test用于存放单元测试代码，src/target用于存放编译、打包后的输出文件。这是全世界maven项目的通用约定，请记住这些固定的目录结构。

3-2. 生成项目

- `mvn idea:idea`
- `mvn eclipse:eclipse`

3-3. 编译源代码

- `mvn compile` 完成编译操作
- `mvn test-compile`

3-4. 清理

- `mvn eclipse:clean`
- `mvn clean`

4.pom的基本结构

4-1.基本顺序

```
<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.lanou</groupId>
    <artifactId>MavenDemo</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>MavenDemo Maven Webapp</name>
    <url>http://maven.apache.org</url>

    <!--定义一些常量值，以供后面进行使用-->
    <properties>
        <testmark>test</testmark>
        <spring.version>4.3.6.RELEASE</spring.version>
        <spring-data.version>1.11.0.RELEASE</spring-data.version>
    </properties>


    <!--依赖关系-->
    <dependencies>
        <!--依赖设置-->
        <dependency>
            <!--依赖组织名称-->
            <groupId>junit</groupId>
            <!--依赖项目名称-->
            <artifactId>junit</artifactId>
            <!--依赖版本名称-->
            <version>4.12</version>
            <!--依赖范围，test包下依赖该设置-->
            <scope>${testmark}</scope>
        </dependency>
    </dependencies>


    <build>
        <finalName>MavenDemo</finalName>
        <resources>
            <!--指定资源目录-->
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                    <include>**/*.tld</include>
                </includes>
```

```
                <filtering>false</filtering>
            </resource>

            <!--指定源代码目录-->
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.properties</include>
                    <include>**/*.xml</include>
                    <include>**/*.tld</include>
                </includes>
                <filtering>false</filtering>
            </resource>

        </resources>

        <plugins>
            <!--编译插件，指定编译用的jdk版本-->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8.0_45</source>
                    <source>1.8.0_45</source>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <port>80</port>
                    <path>/</path>
                </configuration>
            </plugin>

        </plugins>
    </build>
</project>
```

4-2.pom的标签详解

groupId，artifactId和version三行。这三个元素定义了一个项目基本的坐标，在Maven的世界，任何的jar、pom或者war都是以基于这些基本的坐标进行区分的。

- groupId定义了项目属于哪个组，这个组往往和项目所在的组织或公司存在关联,例如`com.google.myapp`(前面代表公司名，后面myapp代表项目名)
- artifactId定义了当前Maven项目在组中唯一的ID，建议使用项目的名称-模块名称例

如：myapp-hello

- version指定了项目当前的版本——1.0-SNAPSHOT。SNAPSHOT意为快照，说明该项目还处于开发中，是不稳定的版本.
- type: 依赖的类型，对应于项目坐标定义的packaging。大部分情况下，该元素不必声明，其默认值是jar
- optional: 标记依赖是否可选
- exclusions: 用来排除传递性依赖，下面会进行详解
- scope: 依赖的范围，下面会进行详解

4-2-1.scope

依赖的范围：三种classpath：

- 编译classpath
- 测试classpath
- 运行classpath
- compile: 编译依赖范围。如果没有指定，就会默认使用该依赖范围。使用此依赖范围的Maven依赖，对于`编译、测试、运行`三种classpath都有效。
- test: 测试依赖范围。使用此依赖范围的Maven依赖，只对于测试classpath有效，在编译主代码或者运行项目的使用时将无法使用此类依赖。典型的例子就是JUnit，它只有在编译测试代码及运行测试的时候才需要。
- provided: 已提供依赖范围。使用此依赖范围的Maven依赖，`对于编译和测试classpath有效，但在运行时无效`。典型的例子是servlet-api，编译和测试项目的时候需要该依赖，但在运行项目的时候，由于容器已经提供，就不需要Maven重复地引入一遍。
- runtime: 运行时依赖范围。使用此依赖范围的Maven依赖，`对于测试和运行classpath有效，但在编译主代码时无效`。典型的例子是JDBC驱动实现，项目主代码的编译只需要JDK提供的JDBC接口，只有在执行测试或者运行项目的时候才需要实现上述接口的具体JDBC驱动。
- system: 系统依赖范围。该依赖与三种classpath的关系，`和provided依赖范围完全一致`。但是，使用system范围依赖时必须通过systemPath元素显式地指定依赖文件的路径。由于此类依赖不是通过Maven仓库解析的，而且往往与本机系统绑定，可能造成构建的不可移植，因此应该谨慎使用。systemPath元素可以引用环境变量，如：

具体如下：

依赖	编译	测试	运行

compile	√	√	×
test	x	√	x
provided	√	√	x
runtime	x	√	√

4-2-2.propreties属性

可以自定义变量, 如下:

```
<properties>
   <testmark>test</testmark>
</properties>

...

<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.12</version>
   <scope>${testmark}</scope>
</dependency>
```