网络教程：

https://zhuanlan.zhihu.com/p/73444114

https://mp.weixin.qq.com/s?src=11&timestamp=1587946858&ver=2303&signature=D4COq2B88HSBHxXw85zpsL*5e*1O6OdfNxgNTwwXTi2jQmnRTcqAbihoqVCuUSfPciIl29mGmq9WyLNDVWs4CrDz24KhkvaNH2YV9IRvXwCUZb1TU17QBIASMpPVYPWm&new=1

一、为什么使用 Maven 这样的构建工具【why】

① 一个项目就是一个工程如果项目非常庞大，就不适合使用 package 来划分模块，最好是每一个模块对应一个工程，利于分工协作。借助于 maven 就可以将一个项目拆分成多个工程

② 项目中使用 jar 包，需要“复制”、“粘贴”项目的 lib 中同样的 jar 包重复的出现在不同的项目工程中，你需要做不停的复制粘贴的重复工作。借助于 maven，可以将 jar 包保存在“仓库”中，不管在哪个项目只要使用引用即可就行。

③ jar 包需要的时候每次都要自己准备好或到官网下载借助于 maven 我们可以使用统一的规范方式下载 jar 包，规范

④ jar 包版本不一致的风险不同的项目在使用 jar 包的时候，有可能会导致各个项目的 jar 包版本不一致，导致未执行错误。借助于 maven，所有的 jar 包都放在“仓库”中，所有的项目都使用仓库的一份 jar 包。

⑤ 一个 jar 包依赖其他的 jar 包需要自己手动的加入到项目中

极大的浪费了我们导入包的时间成本，也极大的增加了学习成本。借助于 maven，它会自动的将依赖的 jar 包导入进来。

二、maven 是什么【what】

① maven 是一款服务于 java 平台的自动化构建工具

② 构建

构建定义：把动态的 Web 工程经过编译得到的编译结果部署到服务器上的整个过程。编译：java 源文件[.java]->编译->Classz 字节码文件[.class]部署：最终在 sevlet 容器中部署的不是动态 web 工程，而是编译后的文件

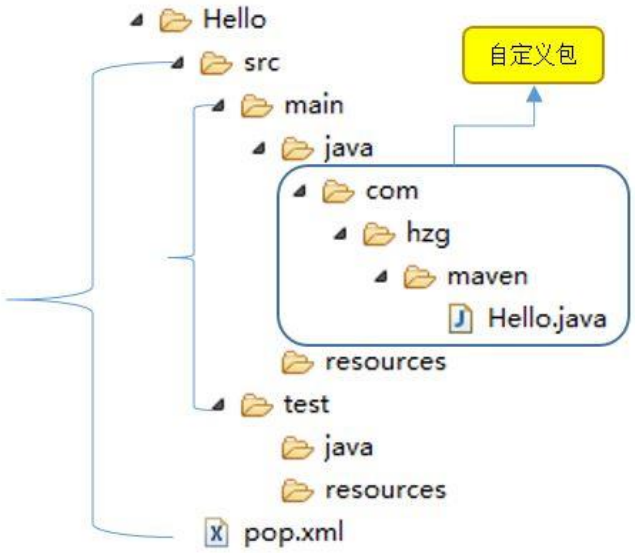
③ 构建的各个环节

- 清理 clean：将以前编译得到的旧文件 class 字节码文件删除
- 编译 compile：将 java 源程序编译成 class 字节码文件
- 测试 test：自动测试，自动调用 junit 程序
- 报告 report：测试程序执行的结果
- 打包 package：动态 Web 工程打 War 包，java 工程打 jar 包
- 安装 install：Maven 特定的概念-----将打包得到的文件复制到“仓库”中的指定位置
- 部署 deploy：将动态 Web 工程生成的 war 包复制到 Servlet 容器下，使其可以运行

四、第一个 maven

① 创建约定的目录结构（maven 工程必须按照约定的目录结构创建）

```
根目录：工程名
|---src：源码
|---|---main:存放主程序
|---|---|---java：java 源码文件
|---|---|---resource：存放框架的配置文件
|---|---test：存放测试程序
|---pop.xml：maven 的核心配置文件
```



② 常用 **maven** 命令

- mvn clean：清理
- mvn compile：编译主程序
- mvn test-compile：编译测试程序
- mvn test：执行测试
- mvn package：打包
- mvn install：安装

执行 maven 命令必须进入到 pom.xml 的目录中进行执行

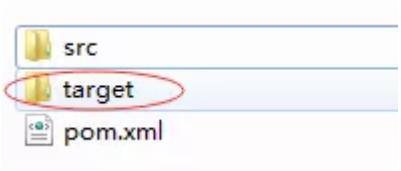
```
C:\Users\hzg>d:

D:\>cd D:\WorkSpace\Hello
```

进入到项目的 pom.xml 目录之后，就可以执行啦。

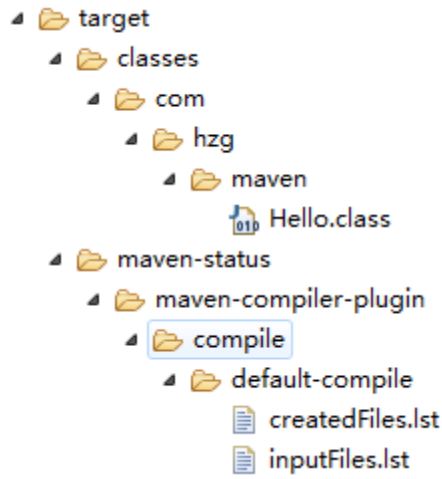
1、运行 mvn compile

```
C:\Users\hzg>d:

D:\>cd D:\WorkSpace\Hello

D:\WorkSpace\Hello>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Hello 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
Downloading: https://repo.maven.apache.org/maven2/junit/junit/4.0/junit-4.0.pom
Downloaded: https://repo.maven.apache.org/maven2/junit/junit/4.0/junit-4.0.pom (
210 B at 0.0 KB/sec)
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Hello ---
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-artifac
t-manager/2.0.6/maven-artifact-manager-2.0.6.jar
Downloading: https://repo.maven.apache.org/maven2/commons-cli/commons-cli/1.0/co
mmons-cli-1.0.jar
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-
descriptor/2.0.6/maven-plugin-descriptor-2.0.6.jar
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-int
eractivity-api/1.0-alpha-4/plexus-interactivity-api-1.0-alpha-4.jar
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-
registry/2.0.6/maven-plugin-registry-2.0.6.jar
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-artifact
-manager/2.0.6/maven-artifact-manager-2.0.6.jar (56 KB at 42.7 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-monitor
/2.0.6/maven-monitor-2.0.6.jar
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-inte
ractivity-api/1.0-alpha-4/plexus-interactivity-api-1.0-alpha-4.jar (14 KB at 8.5
 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/commons-cli/commons-cli/1.0/com
mons-cli-1.0.jar (30 KB at 18.3 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-monitor/
2.0.6/maven-monitor-2.0.6.jar (11 KB at 5.0 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-d
escriptor/2.0.6/maven-plugin-descriptor-2.0.6.jar (37 KB at 9.3 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-plugin-r
egistry/2.0.6/maven-plugin-registry-2.0.6.jar (29 KB at 7.2 KB/sec)
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.e
. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding GBK, i.e. buil
d is platform dependent!
[INFO] Compiling 1 source file to D:\WorkSpace\Hello\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.346 s
[INFO] Finished at: 2017-06-03T23:52:58+08:00
[INFO] Final Memory: 16M/172M
[INFO] ------------------------------------------------------------------------
```

OK，运行完毕，你在 pom.xml 配置的依赖的包已经导入到仓库了，问题来了，**仓库默认的位置在哪？仓库的默认位置：** c:\Usrs[登录当前系统的用户名].m2\repository 刚才执行完 compile 之后，之前的文件夹发生了变化



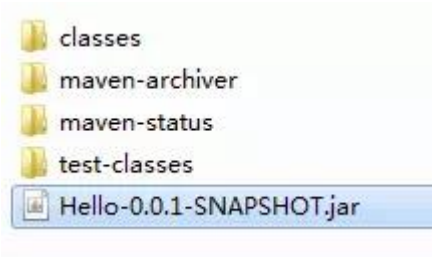
我们发现 Hello 项目里里多了一个 target 文件夹。文件夹的内容为：



发现 target 里主要存放的就是编译后的字节码文件

2、**运行 mvn test-compile**，target 文件夹下面除了 classes 之外多了 test-classes 文件夹

3、**运行 mvn package**，target 文件夹下面又多了一个打好的 jar 包



4、运行 mvn clean，发现整个 target 文件夹都没了。又回到了编译之前我们手动创建的文件夹

五、仓库和坐标

① **pom.xml：** Project Object Model 项目对象模型。它是 maven 的核心配置文件，所有的构

建的配置都在这里设置。

② **坐标：**使用下面的三个向量在仓库中唯一的定位一个 maven 工程

```
<dependencies>
    <dependency>
        <!-- 1、公司或组织域名倒序+项目名 -->
        <groupId>org.springframework</groupId>
        <!-- 2、模块名 -->
        <artifactId>spring-core</artifactId>
        <!-- 3、版本 -->
        <version>4.3.4.RELEASE</version>
        <scope>compile</scope>
    </dependency>
</dependencies>
```

③ **maven 工程的坐标与仓库中路径的关系：**maven 坐标和仓库对应的映射关系：[groupId][artifactId][version][artifactId]-[version].jar 去本地仓库看一下此目录：org\springframework\spring-core\4.3.4.RELEASE\spring-core-4.3.4.RELEASE.jar 果然是完全对应的

④ **仓库**仓库的分类：**1、本地仓库：**当前电脑上的仓库，路径上已经说过了哦 **2、远程仓库：**

- 私服：搭建在局域网中，一般公司都会有私服，私服一般使用 nexus 来搭建。具体搭建过程可以查询其他资料
- 中央仓库：架设在 Internet 上，像刚才的 springframework 就是在中央仓库上

六、依赖

① maven 解析依赖信息时会到本地仓库中取查找被依赖的 jar 包

- 对于本地仓库中没有的会去中央仓库去查找 maven 坐标来获取 jar 包，获取到 jar 之后会下载到本地仓库
- 对于中央仓库也找不到依赖的 jar 包的时候，就会编译失败了

② 如果依赖的是自己或者团队开发的 maven 工程，需要先使用 install 命令把被依赖的 maven 工程的 jar 包导入到本地仓库中

举例：现在我再创建第二个 maven 工程 HelloFriend，其中用到了第一个 Hello 工程里类的 sayHello(String name)方法。我们在给 HelloFriend 项目使用 mvn compile 命令进行编译的时候，会提示缺少依赖 Hello 的 jar 包。怎么办呢？到第一个 maven 工程中执行 mvn install 后，你再去看一下本地仓库，你会发现有了 Hello 项目的 jar 包。一旦本地仓库有了依赖的 maven 工程的 jar 包后，你再到 HelloFriend 项目中使用 mvn compile 命令的时候，可以成功编译**③ 依赖范围**

```
<groupId>org.springframework</groupId>
<artifactId>spring-core</artifactId>
<version>4.3.4.RELEASE</version>
<scope>compile</scope>
```

*scope 就是依赖的范围***1、compile，**默认值，适用于所有阶段（开发、测试、部署、运行），本 jar 会一直存在所有阶段。**2、provided，**只在开发、测试阶段使用，目的是不让 Servlet 容器和你本地仓库的 jar 包冲突 。如 servlet.jar。**3、runtime，**只在运行时使用，如 JDBC 驱动，适用运行和测试阶段。**4、test，**只在测试时使用，用于编译和运行测试代码。不会随项目发布。**5、system，**类似 provided，需要显式提供包含依赖的 jar，Maven 不会在 Repository 中查找它。

七、生命周期

Maven 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”，初学者容易将 Maven 的生命周期看成一个整体，其实不然。这三套生命周期分别是：**① Clean Lifecycle 在进行真正的构建之前进行一些清理工作。**Clean 生命周期一共包含了三个阶段：

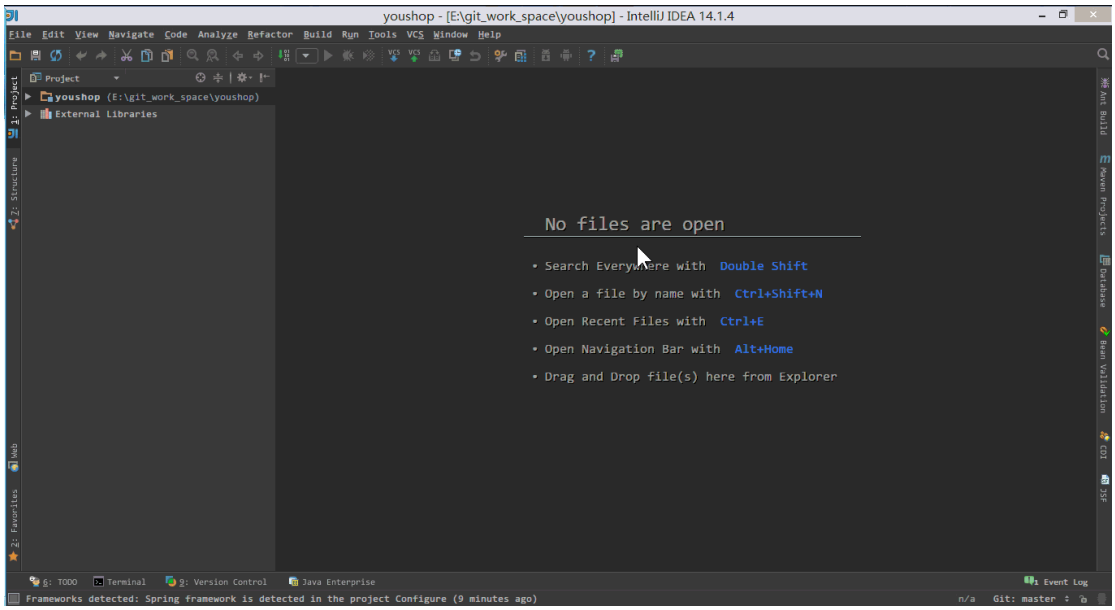
- pre-clean 执行一些需要在 clean 之前完成的工作
- clean 移除所有上一次构建生成的文件
- post-clean 执行一些需要在 clean 之后立刻完成的工作

② Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。

- validate
- generate-sources
- process-sources
- generate-resources
- process-resources 复制并处理资源文件，至目标目录，准备打包
- compile 编译项目的源代码
- process-classes
- generate-test-sources
- process-test-sources
- generate-test-resources
- process-test-resources 复制并处理资源文件，至目标测试目录
- 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
- 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

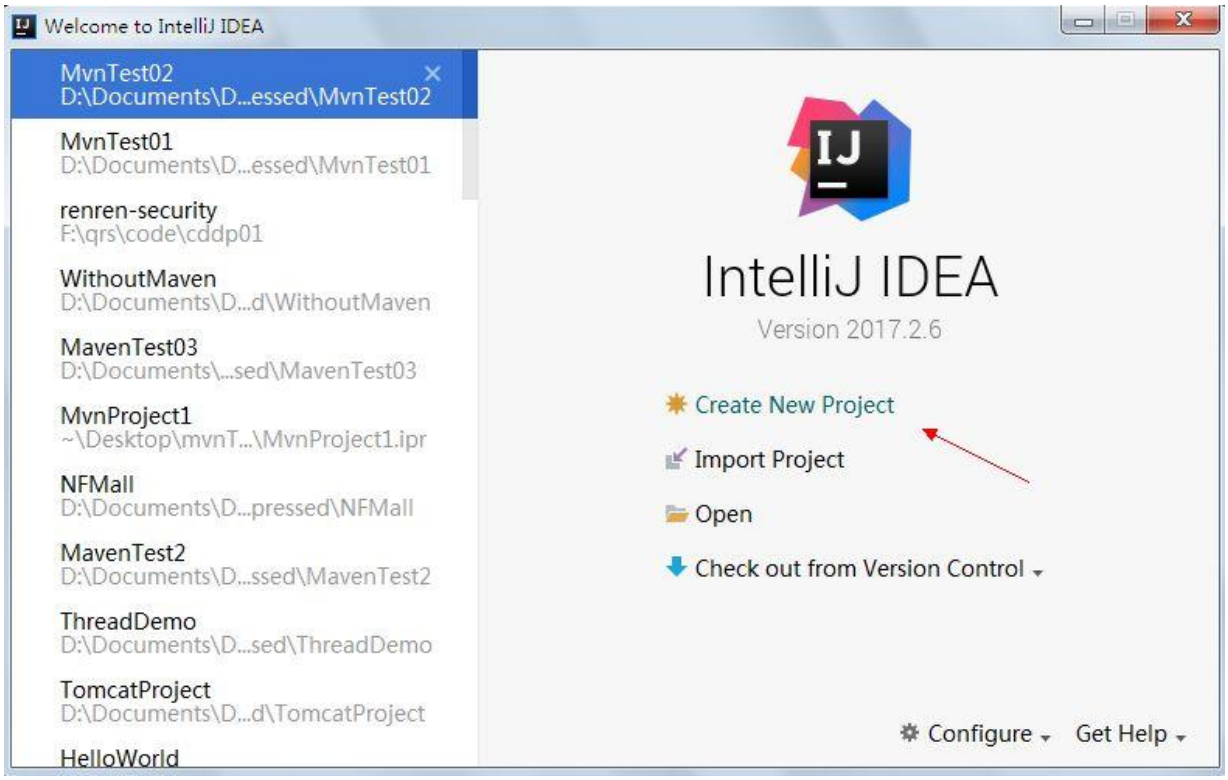
4.2、在 IDEA 中创建 Maven 项目



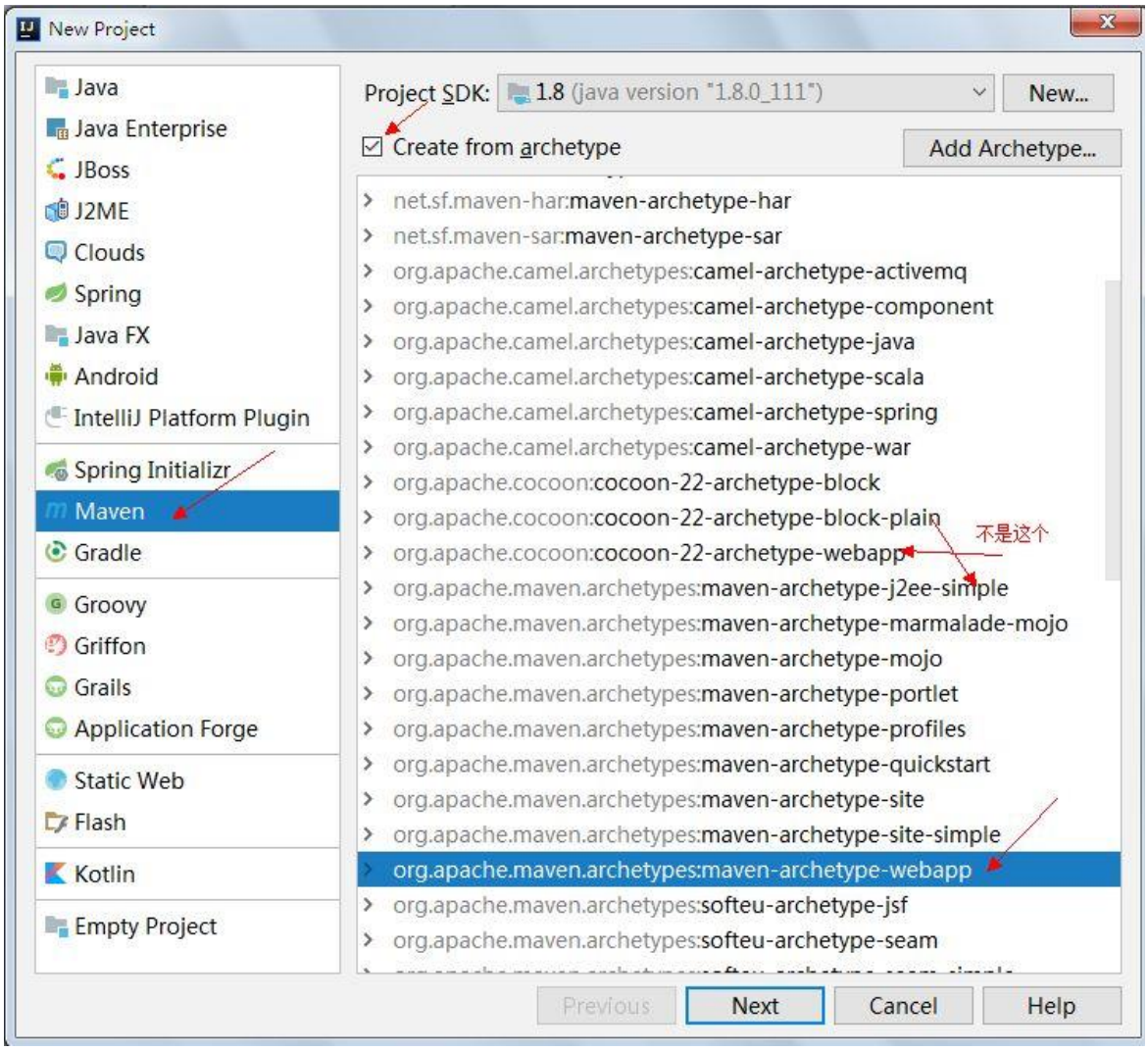
4.2.1、创建项目

4.1 是一种创建 maven 项目的办法，但不推荐，因为没有使用统一的骨架，可以一开始就选择创建 maven 项目，步骤如下：

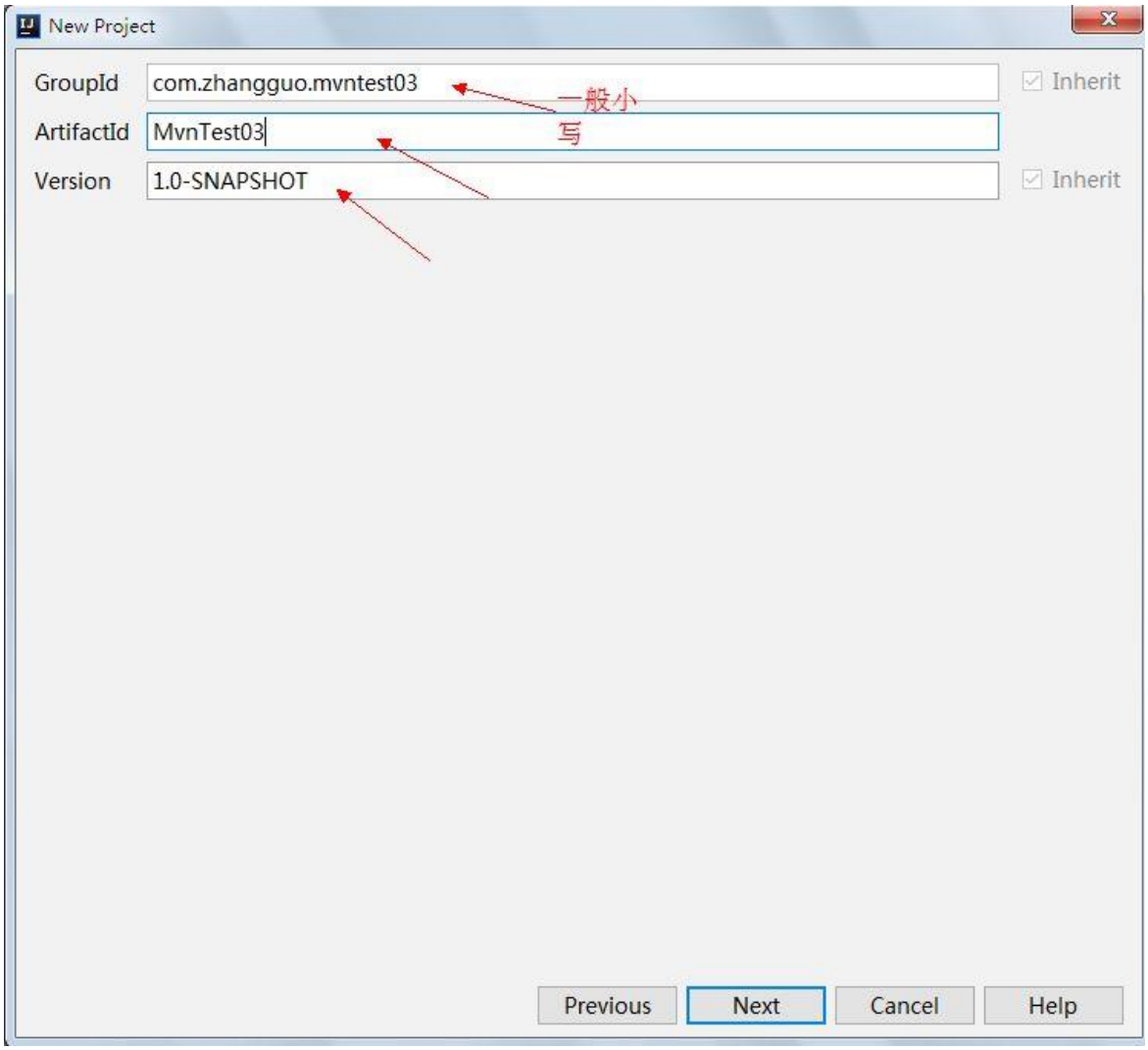
步骤一：首先先创建一个 project,上次我说过了创建一个 project 就是一个工作空间，在这里就是创建一个 maven 的工作空间



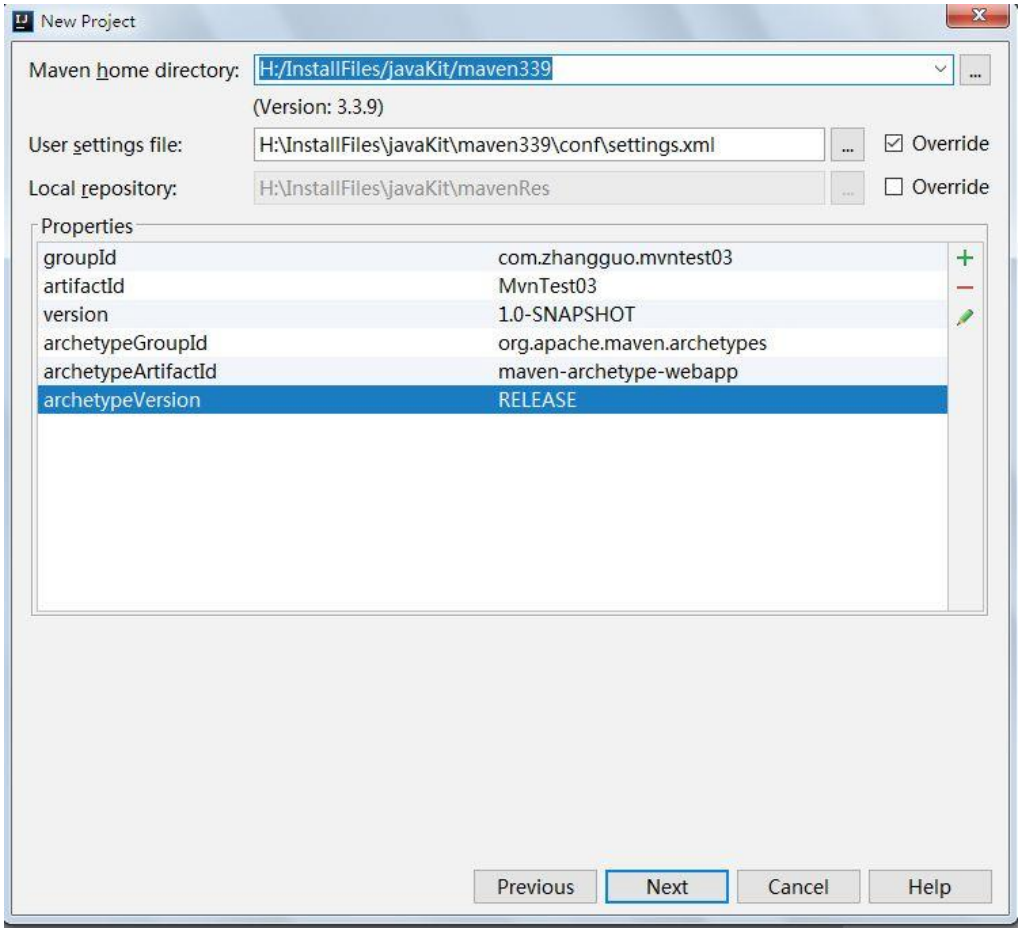
步骤二：选择 maven 项目，指定骨架，这里选择的是一个 webapp，当然 webapp 骨架有非常多，这里选择 apache 提供的



步骤三：填写项目的座标，公司编号（一般倒置域名），项目名称，版本：

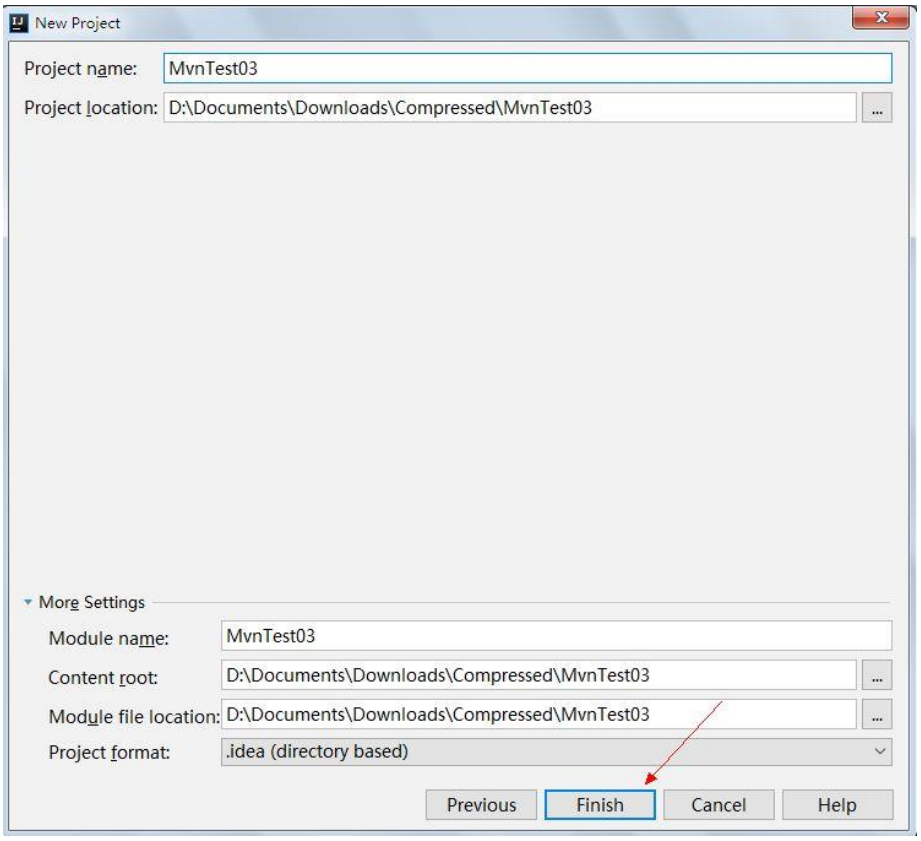


步骤四：因为 IDEA 内置了 Maven，可以选择默认内置的 Maven；当然最好是选择自己安装并配置好的环境，让所有的 IDE 统一：

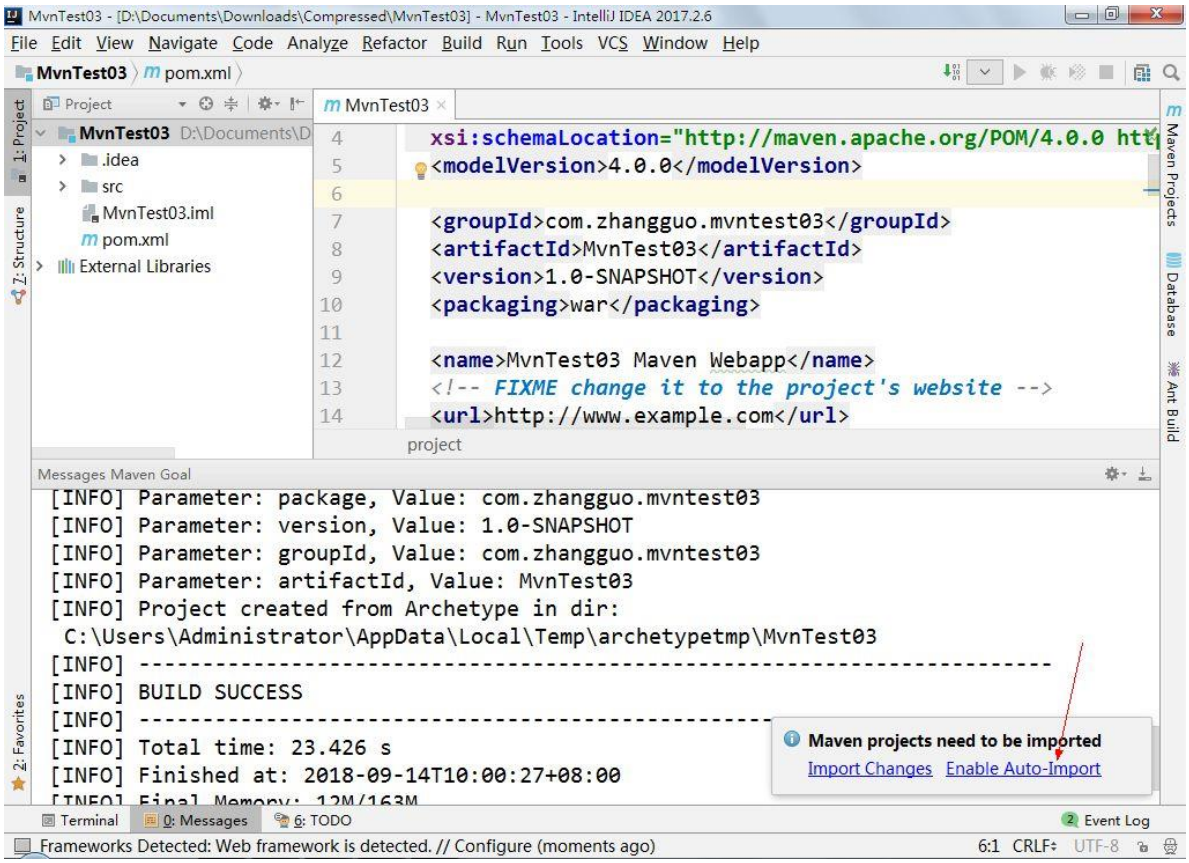


这里可以点绿色的小加号添加参数

步骤五：选择项目名称，位置，一般默认



点击 Finish 项目就创建完成了，如下图所示：


MvnTest03 - [D:\Documents\Downloads\Compressed\MvnTest03] - MvnTest03 - IntelliJ IDEA 2017.2.6
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MvnTest03 pom.xml
Project
MvnTest03 D:\Documents\D
.idea
src
MvnTest03.iml
pom.xml
External Libraries
MvnTest03
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 htt
<modelVersion>4.0.0</modelVersion>
<groupId>com.zhangguo.mvntest03</groupId>
<artifactId>MvnTest03</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
<name>MvnTest03 Maven Webapp</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>
project
Messages Maven Goal
[INFO] Parameter: package, Value: com.zhangguo.mvntest03
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: groupId, Value: com.zhangguo.mvntest03
[INFO] Parameter: artifactId, Value: MvnTest03
[INFO] Project created from Archetype in dir:
C:\Users\Administrator\AppData\Local\Temp\archetypetmp\MvnTest03
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 23.426 s
[INFO] Finished at: 2018-09-14T10:00:27+08:00
Maven projects need to be imported
Import Changes Enable Auto-Import
Terminal 0: Messages 6: TODO Event Log
Frameworks Detected: Web framework is detected. // Configure (moments ago)
6:1 CRLF UTF-8
1: Project 7: Structure 2: Favorites
Maven Projects Database Ant Build