【GeekHour】一小时Maven教程

1. Maven简介

1.1 什么是Maven

Maven 是一个项目管理工具，它可以帮助我们自动化构建项目，管理依赖，发布项目。

1.2 为什么要使用Maven

- **依赖管理**：Java项目中会有很多的依赖库文件，这些库文件可能有很多的依赖关系，如果我们手动去下载这些依赖的话，不但非常的麻烦，而且不同的依赖版本之间可能会有冲突，这个时候就可以使用Maven来帮助我们管理这些依赖，我们需要做的就是在POM文件中告诉Maven我们需要哪些依赖，然后Maven就可以自动的将这个jar包，以及它所依赖的其他所有jar包全部都下载并导入到项目中，非常的方便。
- **构建管理**：在Java项目中，我们需要把Java的源文件编译成字节码文件，然后再把字节码文件打包成一个可执行的jar包或者war包，如果没有一个自动化的构建工具的话，这个过程就会非常的繁琐，而且容易出错，Maven提供了一个标准的项目结构和构建流程，只需要按照这个标准来组织项目，就可以非常轻松方便的构建Java项目。
- **项目管理**：Maven提供了一个标准的项目结构和构建流程，只需要按照这个标准来组织项目，就可以非常轻松方便的构建Java项目。

1.3 Maven的核心概念及工作原理

Maven的核心概念是项目对象模型（Project Object Model，POM），它是一个XML文件，也是 Maven 项目的核心文件，定义了项目的配置、依赖、插件以及构建的过程。Maven读取pom.xml文件之后，会根据这个文件中定义的规则去下载依赖包，然后编译工程中的源代码，最后将工程打包成一个可执行的jar包或者war包，这个过程中会有很多的插件来帮助我们完成这些工作，比如说编译插件、打包插件、测试插件等等，这些插件都是Maven提供的，我们只需要在pom.xml文件中配置一下就可以了，Maven会自动的去执行这些插件，完成构建的过程。

1.4 Maven仓库

Maven中有一个仓库的概念，仓库简单来说就是指存放jar包的地方，按照作用范围的不同可以分为本地仓库、远程仓库和中央仓库。

- 本地仓库就是我们自己电脑上的一个目录，一般默认是在用户家目录(`$HOME`)下的`.m2`这个目录里面，这个位置可以在Maven的配置文件中修改

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.2.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.2.0
https://maven.apache.org/xsd/settings-1.2.0.xsd">
<!--
 | The path to the local repository maven will use to store artifacts.
 | Default: ${user.home}/.m2/repository
-->
<localRepository>/path/to/local/repo</localRepository>
```

- 远程仓库也叫做私服仓库，一般是公司内部搭建的一个仓库，用来给公司内部的项目提供统一的依赖管理，这样就可以避免jar包的重复下载，而且也可以把一些公司内部发布的私有的jar包放到这个仓库里面，供其他项目来使用，一般由公司内部专门的运维人员来维护，最常用的搭建私服仓库的工具是 *Nexus*，远程仓库并不是必须的，如果没有配置的话，Maven会直接去中央仓库中下载依赖。
- 中央仓库是Maven官方提供的一个仓库，里面包含了大量的开源项目，地址是 https://repo.maven.apache.org/maven2

2. 安装配置

2.1 安装配置JDK

Maven要求JDK版本至少在1.7以上，所以首先需要安装JDK，可以到Oracle官网下载JDK，然后配置系统环境变量`JAVA_HOME`和`PATH`。

- Linux/Mac系统

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_291.jdk/Contents/Home
export PATH=$JAVA_HOME/bin:$PATH
```

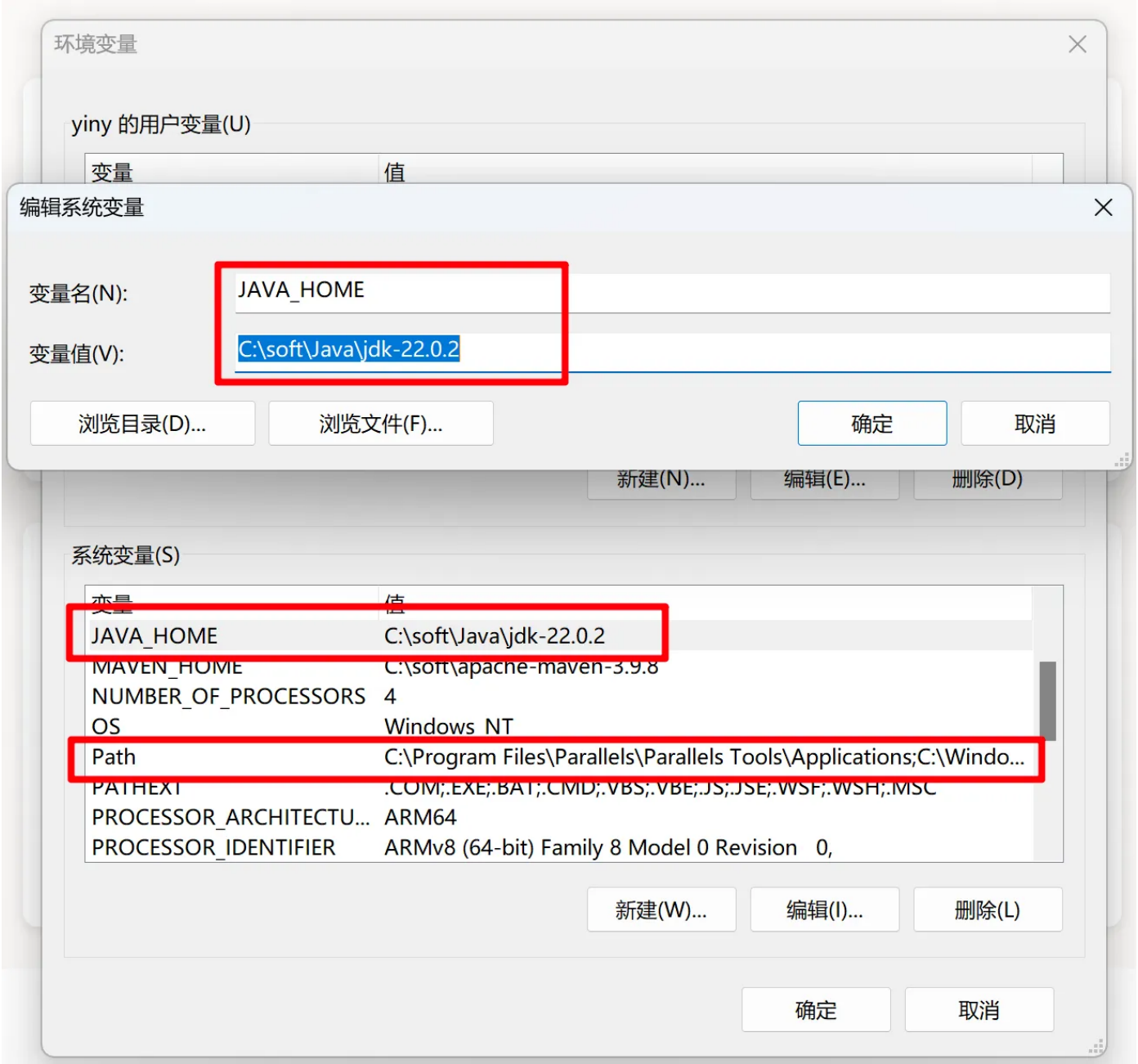
- Windows系统

Windows系统下找到系统环境变量，添加`JAVA_HOME`和`PATH`，如下图所示：

找到系统属性中的环境变量：
`JAVA_HOME`和`PATH`，`JAVA_HOME`的值为`JDK`的安装路径，`PATH`中添加`%JAVA_HOME%\bin`，如下图所示：
如下图所示：


系统属性
计算机名
硬件
高级
系统保护
远程
要进行大多数更改，你必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与登录帐户相关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统故障和调试信息
设置(T)...
环境变量(N)...
确定
取消
应用(A)

设置环境变量：



2.2 下载安装Maven

官网下载地址

配置系统环境变量`MAVEN_HOME`和`PATH`

2.3 系统环境变量配置

2.3.1 Linux/Mac系统

根据使用的`Shell`不同，配置文件也不同，
可以使用`echo $SHELL`来查看当前使用的`Shell`，一般是`bash`或者`zsh`。
在`~/.bashrc`或者`~/.zshrc`文件中添加如下内容：

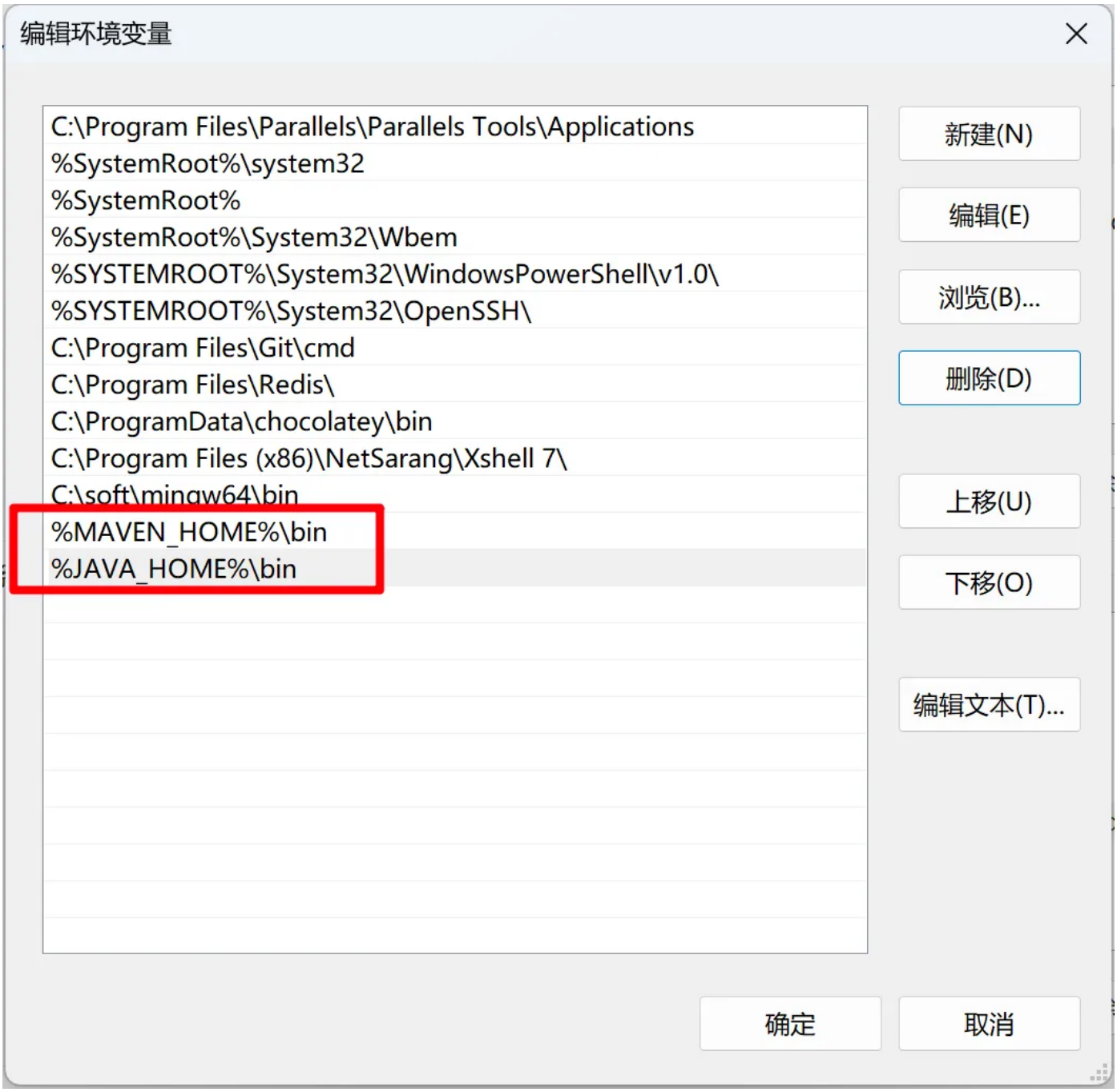
```
export MAVEN_HOME=/Users/yiny/soft/apache-maven-3.9.8
export PATH=$MAVEN_HOME/bin:$PATH
```

配置完成之后需要重启一下终端，或者使用`source ~/.bashrc`或者`source ~/.zshrc`来使配置生效。

2.3.2 Windows 系统

Windows系统下找到系统环境变量，添加`MAVEN_HOME`和`PATH`，如下图所示：

`MAVEN_HOME`的值为Maven的安装路径，`PATH`中添加`%MAVEN_HOME%\bin`，如下图所示：



2.4 配置镜像仓库

由于中央仓库是部署在国外的服务器上，所以下载速度可能会比较慢，我们可以配置一个国内的镜像仓库来加速下载速度，比如阿里云的镜像仓库，配置方法如下：

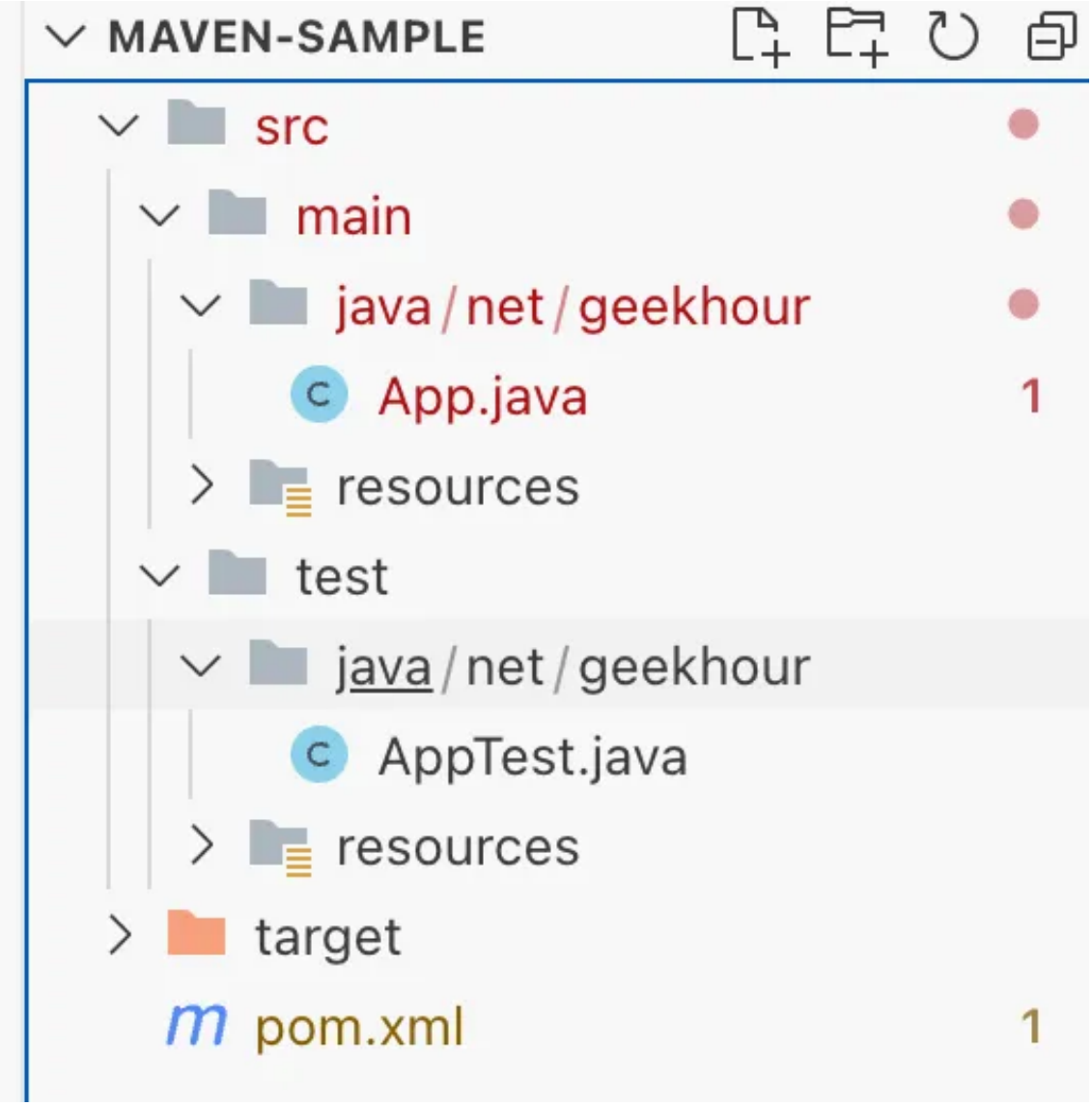
```
<!-- settings.xml -->
<mirrors>
  <mirror>
    <id>aliyunmaven</id>
    <mirrorOf>*</mirrorOf>
    <name>阿里云公共仓库</name>
    <url>https://maven.aliyun.com/repository/public</url>
  </mirror>
</mirrors>
```

3. Maven工程的目录结构

Maven的工程目录结构是有一定的规范的，这样可以方便Maven来自动的构建项目，下面是一个标准的Maven工程目录结构：

```
project                     # 项目根目录
|-- src                     # 源代码目录
|   |-- main                # 主目录
|   |   |-- java            # Java源代码目录
|   |   |-- resources       # 资源文件目录
|   |   |-- webapp          # Web应用目录
|   |-- test                # 测试目录
|       |-- java            # 测试源代码目录
|       |-- resources       # 测试资源文件目录
|-- target                  # 项目构建目录
|-- pom.xml                 # 项目配置文件
```

例如：



4. POM文件

POM文件是Maven项目的核心文件，它是一个XML文件，定义了项目的配置、依赖、插件以及构建的过程。

以下是一个简单的POM文件示例：

```
<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">

  <!-- 模型版本 -->
  <modelVersion>4.0.0</modelVersion>
  <!-- 公司或者组织的唯一标志，并且配置时生成的路径也是由此生成，
       如：com.companyname.project-group，
       maven会将该项目打成的jar包放本地路径：
       /com/companyname/project-group -->
  <groupId>com.companyname.project-group</groupId>

  <!-- 项目的唯一ID，一个groupId下面可能多个项目，就是靠artifactId来区分的 -->
  <artifactId>project</artifactId>

  <!-- 版本号 -->
  <version>1.0</version>

  <!-- 属性变量 -->
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <!-- 依赖 -->
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.3.9</version>
    </dependency>
  </dependencies>

  <!-- 依赖管理 -->
  <dependencyManagement>
    <dependencies>
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-core</artifactId>
          <version>5.3.9</version>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <!-- 仓库管理 -->
  <repositories>
    <repository>
```

```
        <id>central</id>
        <url>https://repo.maven.apache.org/maven2</url>
    </repository>
  </repositories>

  <!-- 构建 -->
  <build>
    <!-- 插件管理 -->
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

5. 构建生命周期

Maven提供了三种主要的生命周期：`Clean`、`Default`和`Site`

5.1 `Clean`：用于项目清理（`mvn clean`）

执行`clean`生命周期，会删除`target`目录下的所有文件，包括编译后的字节码文件、打包后的jar包、生成的站点等等。

```
mvn clean
```

5.2 `Default`：用于项目部署

```
validate` => `compile` => `test` => `package` => `verify` => `install` => `deploy
```

- `mvn clean`：清理项目
- `mvn compile`：编译项目
- `mvn test`：运行测试
- `mvn package`：编译代码并打包成可分发格式，比如jar/war
- `mvn install`：将包安装到本地仓库中，供其他模块使用
- `mvn deploy`：将包部署到远程仓库中，供其他项目和开发人员使用
- `mvn site`：生成项目站点

阶段	处理	描述
`mvn validate`	验证项目	验证项目是否正确且所有必须信息是可用的
`mvn compile`	执行编译	源代码编译在此阶段完成
`mvn test`	测试	使用适当的单元测试框架（例如JUnit）运行测试。
`mvn package`	打包	将编译后的代码打包成可分发的格式，例如 JAR 或 WAR
`mvn verify`	检查	对集成测试的结果进行检查，以保证质量达标
`mvn install`	安装	安装打包的项目到本地仓库，以供其他项目使用
`mvn deploy`	部署	拷贝最终的工程包到远程仓库中，以共享给其他开发人员和工程

5.3 Site：用于生成项目站点

用于生成项目站点，包括项目的文档、报告、API文档等等。

```
# 生成站点文档
mvn site
# 部署站点文档
mvn site:deploy
```

5.4 插件命令

Maven插件扩展了Maven的功能，可以用来完成一些特定的任务，

1. `mvn archetype:generate`：创建一个新的Maven项目，并生成项目骨架
2. `mvn dependency:tree`：查看项目依赖树
3. `mvn dependency:analyze`：分析项目依赖
4. `mvn dependency:resolve`：解析项目依赖
5. `mvn dependency:copy-dependencies`：复制项目依赖
6. `mvn versions:display-dependency-updates`：显示项目依赖的更新

6. 依赖管理

6.1 依赖的范围

- `compile`：默认范围，编译、测试、运行时都有效
- `provided`：编译、测试有效，运行时无效，比如servlet-api
- `runtime`：测试、运行时有效，编译时无效
- `test`：测试时有效，编译、运行时无效
- `system`：类似`provided`，但是需要指定jar包的路径
- `import`：导入依赖的范围

6.2 依赖的传递性

Maven会自动的解决依赖的传递性，比如说A依赖B，B依赖C，那么Maven会自动的将C也导入到A中，这样就不需要我们手动的去导入C了。
只有当依赖的范围是`compile`或者`runtime`的时候，依赖才会被传递，如果依赖的范围是`provided`或者`test`的时候，依赖是不会被传递的。

6.3 依赖的排除

有时候我们引入的依赖包中可能会包含一些我们不需要的依赖，
这个时候我们可以使用`<exclusions>`标签来排除这些依赖。

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.1.11</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

也可以通过`<optional>`标签来指定依赖是否可选，如果依赖是可选的，那么在引入这个依赖的时候，可以不用引入这个依赖的依赖。

6.4 依赖的版本冲突

当通过依赖传递导入的两个依赖包版本不一致时，Maven会根据一定的规则来解决这个冲突，
一个是最短路径优先，另一个是先声明优先。

7. Nexus私服仓库

Nexus是最常用的Maven私服仓库，可以用来存放公司内部的jar包，以及一些第三方的jar包，这样就可以避免重复下载，提高构建速度。

7.1 下载安装Nexus

下载地址：https://help.sonatype.com/en/download.html

7.2 安装配置Nexus

7.2.1 直接解压安装

下载完成之后解压到本地，然后进入到解压后的目录，执行`bin/nexus run`命令，启动Nexus服务。

Linux或者Mac环境可以直接执行`bin/nexus run`命令来启动，
Windows环境可以执行`bin/nexus.bat run`命令来启动，
启动之后可以通过浏览器访问`http://localhost:8081`来访问Nexus的管理界面，
第一次登录会提示输入用户名和密码，
用户名默认是admin，
密码可以在`nexus-data/admin.password`文件中查看。

7.2.1 使用Docker安装

- Mac (Apple Silicon)

```
docker pull klo2k/nexus3
docker run -d -p 8081:8081 --name nexus klo2k/nexus3
```

- Windows 和其他系统

```
docker run -d -p 8081:8081 --name nexus sonatype/nexus3
```

7.3 登录Nexus

第一次登录需要到配置文件中修改默认密码

```
docker exec -it nexus cat /nexus-data/admin.password
```

登录后修改即可

7.4 创建和管理仓库

登录之后，可以在左侧的`Repositories`菜单中创建仓库，
一般会创建四个仓库：

- `releases`：用来存放正式版本的jar包
- `snapshots`：用来存放快照版本的jar包
- `proxy`：代理中央仓库，用来缓存中央仓库的jar包
- `public`：用来发布jar包，组合了以上三种仓库

7.5 配置连接私服仓库

7.5.1 修改Maven的settings.xml文件

修改`settings.xml`，配置私服仓库地址，
使得Maven可以从私服仓库中下载jar包。

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

如果Nexus中不允许匿名访问，需要在`settings.xml`中配置用户名和密码

```
<servers>
  <server>
    <id>maven-nexus</id>
    <username>admin</username>
    <password>admin</password>
  </server>
```

7.5.2 修改项目的pom.xml配置

修改项目中的`pom.xml`文件，配置私服仓库地址，
使项目可以从私服仓库中下载jar包，或者上传jar包到私服仓库中。

```
<!-- 发布管理 -->
<distributionManagement>
  <!-- 正式版本 -->
  <repository>
    <id>maven-nexus</id>
    <name>Project Releases Repositories</name>
    <url>http://localhost:8081/repository/maven-releases/</url>
  </repository>
  <!-- 快照版本 -->
  <snapshotRepository>
    <id>maven-nexus</id>
    <name>Project Snapshots Repositories</name>
    <url>http://localhost:8081/repository/maven-snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

注意：这里的id要和settings.xml中的id一致

7.6 上传jar包到私服仓库

执行一个`mvn deploy`命令，就可以将jar包上传到私服仓库中，
上传之后在Nexus的管理界面中就可以看到对应的jar包。