Maven是服务于Java平台的自动化构建工具，Maven本身也是Java语言编写的

构建过程中的各个环节：

1.清理：删除以前的编译结果，为重新编译做好准备。

2.编译：将Java源程序编译为字节码文件。

3.测试：针对项目中的关键点进行测试，确保项目在迭代开发过程中关键点的正确性。

4.报告：在每一次测试后以标准的格式记录和展示测试结果。

5.打包：将一个包含诸多文件的工程封装为一个压缩文件用于安装或部署。Java工程对应jar包，Web工程对应war包。

6.安装：在Maven环境下特指将打包的结果——jar包或war包安装到本地仓库中。

7.部署：将打包的结果部署到远程仓库或将war包部署到服务器上运行。

Maven工程手动创建需要创建约定的目录结构：

1.根目录：工程名

2.src目录：源码目录

3.pom.xml文件：Maven工程的核心配置文件POM（Project Object Model）项目对象模型

4.main目录：主程序文件目录

5.test目录：测试程序文件目录

6.Java目录：Java源文件目录

7.resources目录：框架或其他配置文件目录

框架使用一般通过两种方式：以配置文件的方式即（xml文件）或按照框架内部默认约定，一般约定>配置>编码

Maven工程的配置文件pom.xml一般格式：

<?xml version="1.0" ?>

<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">

<modelVersion>4.0.0</modelVersion>（pom文件的结构模型版本）

<groupId>一般为公司或组织的域名倒序+当前项目名称</groupId>（Maven工程坐标，工程坐标与仓库中路径是对应的，对应关系为：groupId标签值/ artifactId标签值/ version标签值/ artifactId标签值-version标签值.jar，groupId标签值与 artifactId标签值中的点在仓库路径中换做/）

<artifactId>当前项目的模块名称</artifactId>

<version>当前模块的版本</version>（SNAPSHOT快照版不稳定版开发版，RELEASE分离版稳定版正式版）

<packaging>打包方式（Maven父工程一般为pom，子工程一般不设置）</packaging>

<name>Maven工程名称</name>（工程名称）

<parent>（Maven子工程引用Maven父工程，引用完成后，可以将Maven子工程里相同的Maven工程坐标删除）

<groupId>Maven父工程工程坐标</groupId>

<artifactId> Maven父工程工程坐标</artifactId>

<version> Maven父工程工程坐标</version>

<relativePath> Maven父工程的pom.xml文件的相对路径</relativePath>（以当前pom.xml文件为基准的Maven父工程的pom.xml文件位置的相对路径）

</parent>

<dependencyManagement>（配置依赖的管理，Maven父工程里配置，子工程里）

<dependencies>

<dependency>

<groupId> Maven父工程依赖的工程坐标</groupId>

<artifactId> Maven父工程依赖的工程坐标</artifactId>

<version> Maven父工程依赖的工程版本号</version>

<scope> Maven父工程依赖的工程范围</scope

</dependency>

</dependencies>

</dependencyManagement>

<modules>（Maven总工程配置聚合工程，Maven总工程安装完成后会按照依赖的关系依次安装相应的聚合工程）

[<module>聚合工程的相对路径</module>]

</modules>

<properties>（声明pom.xml内部属性，通过${属性名}来调用属性值）

<属性名>属性值</属性名>

</properties>

<dependencies>（工程依赖的Maven工程，依赖的原则是验证路径最短者优先，验证路径相同在dependencies标签先声明者优先）

<dependency>

<groupId>依赖的Maven工程坐标</groupId>

<artifactId>依赖的Maven工程名称</artifactId>

<version>依赖的Maven工程的版本号</version>（继承了Maven父工程的子工程可以选择删除版本号使用父工程统一的版本号）

<scope>依赖的常用范围compile（主程序测试程序可访问，具有依赖传递性，参与打包部署），test（主程序不可访问，测试程序可访问，不具有依赖传递性，不参与打包部署），provided（主程序测试程序可访问，不具有依赖传递性，不参与打包部署），依赖范围设置错误会出现依赖冲突，可能报空指针异常</scope>

<exclusions>（排除不需要的依赖）

<exclusion>

<groupId>需要排除的依赖的Maven工程坐标</groupId>

<artifactId>需要排除的依赖的Maven工程名称</artifactId>

</ exclusion >

</ exclusions >

</dependency>

</dependencies>

<build>（配置当前Maven工程构建过程中的特殊配置）

<finalName>名称</finalName>（Maven工程最终的名称）

<plugins>（配置构建过程中需要使用的插件）

<plugin>

<groupId>Maven工程插件的工程坐标</groupId>

<artifactId> Maven工程插件的工程坐标</artifactId>

<version> Maven工程插件的工程坐标</version>

<configuration>（对应Maven工程的插件配置）

<container>（配置当前系统中对应插件的位置）

<containerId>插件名称</containerId>

<home>插件绝对路径</home>

</container>

<configuration>

<type>existing</type>

<home>插件绝对路径</home>

<properties>（配置插件参数属性）

<cargo.servlet.port>属性值（如果Tomcat服务器端口号默认是8080可以不要配置）</cargo.servlet.port>

</properties>

</configuration>

</configuration>

<executions>（配置插件执行情况）

<execution>

<id>cargo-run</id>

<phase>Maven工程生命周期阶段</phase>（指定插件在哪个阶段开始执行）

<goals>

<goal>目标</goal>（指定插件的执行目标）

</goals>

</execution>

</executions>

</plugin>

<!--使用Maven插件进行打包-->

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<configuration>

<archive>

<manifest>

<mainClass></mainClass>

</manifest>

</archive>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

</configuration>

</plugin>

</plugins>

</build>

</project>

Maven执行与构建过程相关的命令需要在pom.xml文件所在目录下执行

Maven常用的构建过程命令：

1.mvn clean清理

2.mvn compile编译主程序

3.mvn test-compile编译测试程序

4.mvn test执行测试

5.mvn package打包

6.mvn install安装，将自定义开发的Maven工程安装到本地仓库中

7.mvn site生成站点

8.mvn clean install site执行Maven所有生命周期即Maven所有操作

Maven执行命令时需要用到其他插件（即jar包）首先到本地仓库（默认位置：当前系统用户家目录\.m2\repository）查找，如果没有找到相应的插件，Maven会自动连接网络到中央仓库去下载，如果不能连接网络，则构建失败

修改指定Maven本地仓库位置：在Maven的配置文件：conf\settings.xml中的根标签settings下添加标签<localRepository>指定的本地仓库路径</localRepository>

Maven自带插件下载失败一般需要手动指定镜像，在Maven的配置文件：conf\settings.xml中的标签mirrors加入镜像子标签：

<mirror>（Maven官方镜像）

<id>Central</id>

<mirrorOf>central</mirrorOf>

<name>Human Readable Name for this Mirror.</name>

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

</mirror>

<mirror>（阿里云镜像）

<id>nexus-aliyun</id>

<mirrorOf>\*</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

Maven仓库分类：

本地仓库：当前电脑上部署的仓库目录，为当前电脑上的所有Maven工程服务

远程仓库：

1.私服（Nexus）：搭建在局域网环境中，为局域网范围内的所有Maven工程服务

2.中央仓库：架设在Internet上，为全世界所有Maven工程服务

3.中央仓库镜像：为了分担中央仓库的流量，提升用户访问速度

Maven仓库中保存的内容是Maven工程，包含Maven自身需要的的插件、第三方框架或者工具的jar包以及自定义开发的Maven工程

Maven解析依赖信息是会到本地仓库中查找被依赖的jar包，自定义开发的Maven工程，需要mvn install安装到本地仓库中

Maven的生命周期：Maven生命周期定义了各个构建环节的执行顺序，Maven的每一个命令都是从生命周期开始阶段执行到指定命令位置阶段。

Maven有三套相互独立的生命周期,分别是：

Clean Lifecycle在进行真正的构建之前进行一些清理工作，包含三个阶段：

pre-clean执行一些需要在cean之前完成的工作

cean移除所有上一次构建生成的文件

post-clean执行一些需要在cean之后立刻完成的工作

Default Lifecycle构建的核心部分，编译，测试，打包，安装，部署等等，包含阶段：

validate

generate-sources

process-Sources+

generate-resources

process-resources复制并处理资源文件，至目标目录，准备打包

compile编译项目的源代码

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-packag

ackage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或部署到服务器上运行

Site Lifecycle生成项目报告，站点，发布站点，包含四个阶段

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

Maven生命周期的每个阶段对应的目标是由相应插件完成的

Maven工程配置继承后，执行安装命令需要先安装父工程

启动Sevlet容器的组织网址https://codehaus-cargo.github.io/cargo/Home.html

Maven工程pom.xml中依赖信息的查找网址http://mvnrepository.com/