Maven基础

Maven概念

Maven能干啥?

1. Maven可以管理jar文件
2. 自动下载jar和他的文档，源代码
3. 管理jar直接的依赖，自动下载依赖所需的jar包
4. 管理你需要的jar版本
5. 编译程序，把Java编译为class
6. 测试代码是否争取
7. 打包文件，形成jar文件或者war文件
8. 部署项目

构建：项目的构建

构建是面向过程的，就是一些步骤，完成项目代码的编译，测试，运行，打包，部署等等。

Maven支持的构建包括有：

1. **清理**，把之前项目编译的东西删除掉，为的编译代码做准备。
2. **编译**，把程序源代码编译为执行代码，java-class文件。maven可以批量的把多个文件编译为class，javac一次只能编译一个文件。
3. **测试**，maven可以执行测试程序代码，验证功能是否正确。批量的测试多个文件。
4. **报告**，生成测试结果的文件，测试通过没有。
5. **打包**，把项目所有的class文件等所有资源放到一个压缩文件中，该文件就是项目的结果文件，即Java程序。压缩文件是jar扩展名的。对于web应用，压缩文件扩展名是.war。

6. **安装**，把5中生成的文件jar,war安装到本机仓库。
7. **部署**，把程序安装好可以执行。

Maven核心概念

POM

一个文件，名称是pom.xml，项目对象模型。maven把一个项目当作一个模型使用。控制maven构建项目的过程，管理jar依赖。

约定的目录结构

maven项目的目录和文件的位置都是规定的

坐标

一个唯一的字符串，用来表示资源的

依赖管理

管理你的项目可以使用jar文件

仓库管理（了解）

你的资源存放的位置

生命周期（了解）

maven工具项目构建的过程，就是生命周期。

插件和目标（了解）

执行maven构建的时候用的工具就是插件

继承

聚合

安装和配置

1. 需要从maven的官网下载maven的安装包apache-naven-3.3.9-bin.zip
2. 解压安装包,解压到一个目录，非中文目录

   子目录：

   1. bin:执行程序，主要是mvn.cmd
   2. conf：maven工具本身的配置文件，settings.xml
3. 配置环境变量

   在系统的环境变量中，指定一个M2_HOME的名称，指定它的值是maven工具安装目录，bin之前的目录--E:\apache-maven-3.3.9，在path加上%M2_HOME%\bin
4. 验证，mvn-v

约定的目录结构

maven约定的目录结构，约定是大家都遵循的一个规则

每一个maven项目在磁盘中都是一个文件夹（项目-hello）

Hello/

---/src

---/main #放你的主程序 java代码和配置文件

---/java #你的程序包和包中的java文件

---/resoures #你的Java程序中要使用的配置文件

---/test #放测试程序代码和文件的

---/java #测试程序包和包中的java文件

---/resoures #测试Java程序中要使用的配置文件

---/pom.xml #maven的核心文件（maven项目必须有）

设置本机存放资源的目录位置

修改maven的配置文件，maven安装目录/conf/settings.xml。先备份settings.xml

修改<local_reository>指定你的目录 </local_reository>

仓库

仓库是什么

仓库是存放东西的，存放maven使用的jar和我们项目使用的jar（第三方工具）

仓库分类

本地仓库：放在本地计算机上的文件夹，存放各种jar

远程仓库：放在互联网上的，使用网络才能使用的仓库。中央仓库：最权威，所有开发人员共享的一个集中仓库；中央仓库的镜像：中央仓库的备份；私服：在公司内部局域网中使用

仓库的使用

maven仓库的使用不需要人为参与。

开发人员需要使用mysql驱动--》maven首先查看本地仓库--》私服--》镜像--》中央仓库

Pom

项目对象模型

```
//groupId 、modelVersion、version 是合三为一即坐标，唯一值，在互联网中唯一标识一个项目的。
<groupId>com.qiangliu8</groupId>
<artifactId>ch01-maven</artifactId>
<version>1.0-SNAPSHOT</version>
```

参数	
modelVersion	maven模型的版本
groupId	组织id，一般是公司域名的倒写。com.qiangliu8.maven
artifactId	项目名称，也是模块名称
version	项目的版本号，-SNAPSHOT表示最新版本

www.mvnrepository.com: 搜索使用的中央仓库，使用groupId或者artifactId作为搜索条件

参数	
packaging	打包后压缩文件的扩展名，默认是jar，可以是war(web应用)
dependencies和dependency	你的项目中要使用的各种资源说明,相当用java的import

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <!-- 依赖 -->
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.23</version>
</dependency>
```

参数	
properties	设置属性
build	表示与构建相关的配置，例如设置编译插件的jdk版本

maven生命周期，常用命令，插件

maven的生命周期：就是maven构建项目的过程，清理，编译，测试，报告，打包，安装。部署。

maven的命令：maven独立使用，通过命令完成maven的生命周期的执行。

maven的插件：maven命令执行时，真正完成功能的插件，就是一些jar文件。

单元测试：用的时junit,junit时一个专门测试的框架工具。测试的内容：测试的时类中的方法，每一个方法都是独立测试的。方法时测试的基本单位（单元）。

maven借助单元测试，批量的测试你类中的大量的方法是否符合预期的。

使用步骤:

1. 加入依赖，在pom.xml加入单元测试依赖

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

2. 在maven的src/test/java目录下，创建测试程序。

   名称：Test+类名

   方法：Test+方法名

3. 方法是public，必须的。

   方法没有返回值，必须的。

   方法上加上注解@Test

命令

mvn clean	清理
mvn compile	编译主程序，生成一个target目录
mvn test compile	编译测试程序，生命一个target目录
mvn test	测试，生命一个目录surefire-reports,保存测试结果
mvn package	会编译、编译测试、测试、并且按照 pom.xml配置把主程序打包生成jar包或者war包
mvn install	安装主程序(会把本工程打包，并且按照本工程的坐标保存到本地仓库中)
mvn deploy	部署主程序(会把本工程打包，按照本工程的坐标保存到本地库中，并且还会保存到私服仓库中。还会自动把项目部署到web容器中)。

mvn compile

编译main/java/目录下的java为class文件，同时把class拷贝到target/classes目录下面

把main/resources目录下的所有文件都拷贝到target/classes目录下

常用插件

控制配置 maven构建项目的参数配置，设置jdk的版本

```
<!--  控制配置 maven构建项目的参数配置，设置jdk的版本-->
  <build>
<!--    配置插件-->
    <plugins>
<!--      配置具体的插件-->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
<!--          插件的名称-->
        <artifactId>maven-site-plugin</artifactId>
<!--          插件的版本-->
        <version>3.3</version>
<!--          配置插件的信息-->
        <configuration>
<!--&lt;!&ndash;          告诉maven，我们写的代码时在jdk1.8上编译的&ndash;&gt;-->
<!--          <source>1.8</source>-->
```

```
<!--&lt;!&ndash;          我们程序运行在jdk1.8上jdk的&ndash;&gt;-->
<!--          <target>1.8</target>-->
        </configuration>
      </plugin>
    </plugins>
  </build>
```

```
//代码时在jdk15上编译运行
<properties>
  <java.version>15</java.version>
  <maven.compiler.source>${java.version}</maven.compiler.source>
  <maven.compiler.target>${java.version}</maven.compiler.target>
</properties>
```

maven在idea中的设置

在idea中设置maven , 让idea和maven结合使用。idea中内置了maven ， 一般不使用内置的，因为用内置修改maven的设置不方便。使用自己安装的maven，需要覆盖idea中的默认的设置。让idea指定maven安装位置等信息。

配置的入口①:配置当前工程的设置，file--settings ---Build,Excution ,Deployment--Build

--Maven

Maven Home directory : maven的安装目录

User settings File :就是maven安装目录

conf/setting. xml配置文件

Local Repository :本机仓库的目录位置

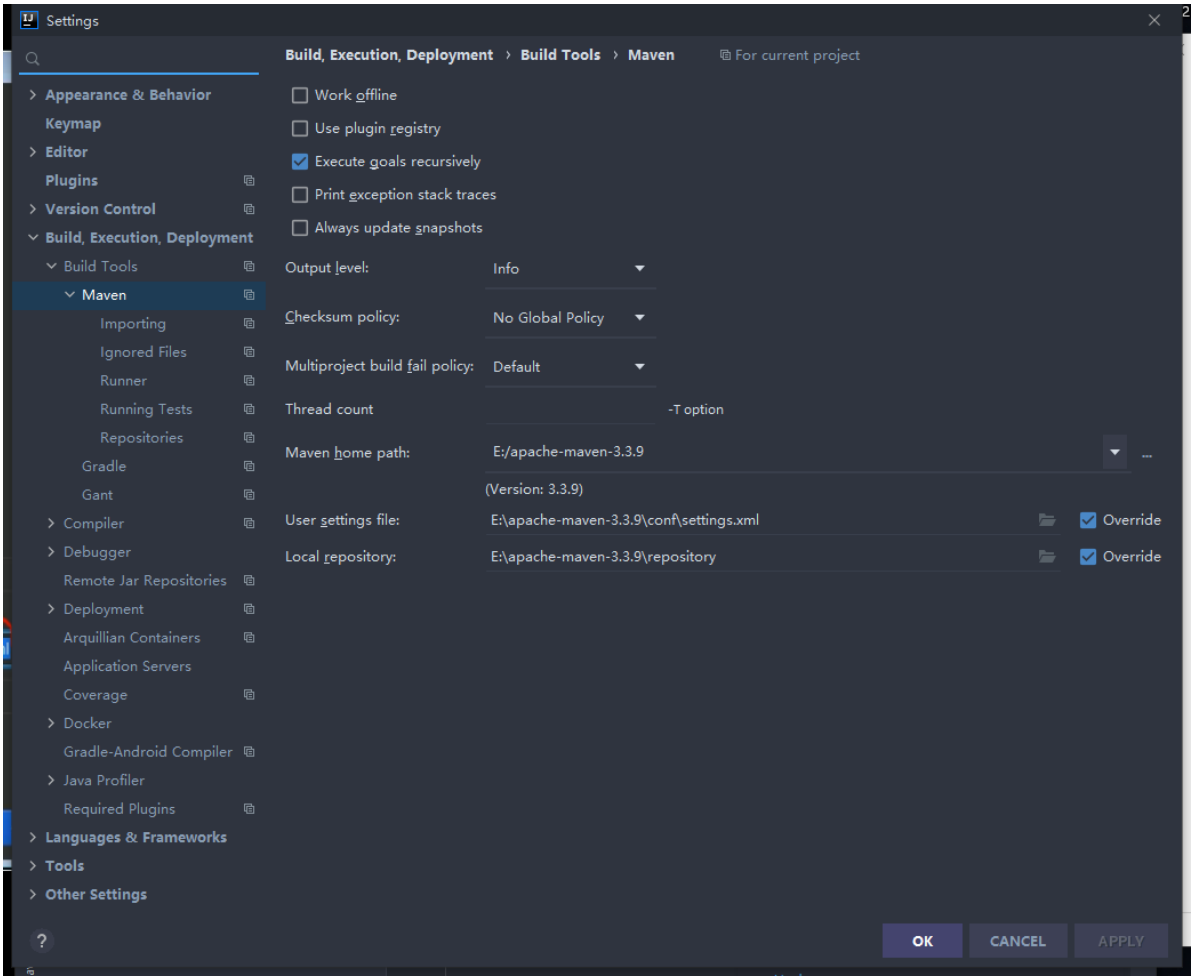
--Build Tools--Maven--Runner

VM Options : -Darchetypecatalog-internal

JRE:你项目的jdk

archetypeCatalog=internal , maven项目创建时，会联网下载模版文件，比较大，使用archetypecatalog-internal，不用下载，创建maven项目速度快

配置以后新建工程的设置，file--New Project settings--settings for New Project



依赖范围

使用scope表示，值由compile,test,provided.

scope:表示以来使用的范围，也就是在maven构建项目的那些阶段(编译,测试,打包,安装,部署)中起作用。默认是compile.表示所有阶段都会用. provide表示编译测试时需要.

```
比如junit的依赖范围是test,只有编译时参与工具.
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

依赖的范围：compile、test、provided，默认采用 compile

	compile	test	provided
对主程序是否有效	是	否	是
对测试程序是否有效	是	是	是
是否参与打包	是	否	否
是否参与部署	是	否	否

maven常用设置

避免中文乱码

```
<properties>
 <maven.compiler.source>1.8</maven.compiler.source> 源码编译 jdk 版本
 <maven.compiler.target>1.8</maven.compiler.target> 运行代码的 jdk 版本
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> 项目构建使用的
编码，避免中文乱码
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding> 生成报
告的编码
 </properties>
```

在 Maven 的 pom.xml 文件中，用于定义全局变量，POM 中通过${property_name}的形式引用变量的值。

定义全局变量：

```
<properties>
 <spring.version>4.3.10.RELEASE</spring.version>
</properties>
```

引用全局变量：

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
```

资源文件:

默认没有使用resources的时候,maven执行编译代码时,会把src/main/resources目录中的文件拷贝到target/classes目录中.

对于src/main/java目录下的非java文件不处理，不拷贝到target/classes目录中

```
<build>
<resources>
 <resource>
 <directory>src/main/java</directory><!--所在的目录-->
 <includes><!--包括目录下的.properties,.xml 文件都会扫描到-->
 <include>**/*.properties</include>
 <include>**/*.xml</include>
 </includes>
 <!-filtering 选项 false 不启用过滤器， *.property 已经起到过滤的作用了 -->
 <filtering>false</filtering>
 </resource>
 </resources>
</build>
```