# PLUS CONTONT

## CLASSPATH

叫做类路径，.class文件的路径，JVM寻找要运行的字节码文件需要依靠它，要是没找到再去c:\test目录找参数后跟着的class文件并执行，如果出现两个同样的文件，JVM选择执行最先找到的文件

## jar包

jar包可以把package组织的目录层级，以及各个目录下的所有文件（包括.class文件和其他文件）都打成一个jar文件，jar包相当于一个zip的压缩文件，我们可以看到目录，如果要执行包里一个class文件，我们可以把它放在classpath里，JVM就会自动在包内寻找并执行。在创建jar包时我们可以直接压缩并改后缀名。/META-INF/MANIFEST.M/META-INF/MANIFEST.MF是一个纯文本文件，提供jar包的信息，JVM会自动读取它，我们可以通过它在cmd输入更方便的命令，对于建立jar包而言，它不是必需的，但如果打算执行，或者要包含一些配置信息，那么建立一个是很有必要的

## 创建jar包

详情参见同文件架中的jar包 运行截图：

```
已添加清单
正在添加: com/(输入 = 0) (输出 = 0)(存储了 0%)
正在添加: com/ISEKAI/(输入 = 0) (输出 = 0)(存储了 0%)
正在添加: com/ISEKAI/HelloWorld.class(输入 = 321) (输出 = 240)(压缩了 25%)
正在添加: com/ISEKAI/Test.class(输入 = 340) (输出 = 250)(压缩了 26%)
正在添加: com/ISEKAI/tool/(输入 = 0) (输出 = 0)(存储了 0%)
正在添加: com/ISEKAI/tool/Print.class(输入 = 389) (输出 = 274)(压缩了 29%)

C:\Users\BEE172\Desktop\微光招新\2024090904015-王勇-JAVA-02\task2\out>
```

利用cmd打包

```
C:\Users\BEE172\Desktop\微光招新\2024090904015-王勇-JAVA-02\task2\out>java -jar A.jar
Hello World
```

用cmd检验打包是否成功，采用vscode和winrar来修改清单文件

## 包的依赖可能遇到的问题

版本不对，比如a包依赖了b包和c包，说b包需要c包的1.0版本才能正常运行，但a包依赖的是c包的1.1版本（该版本删去了1.0中支持b包运行的部分），导致产生了冲突

## maven搭建

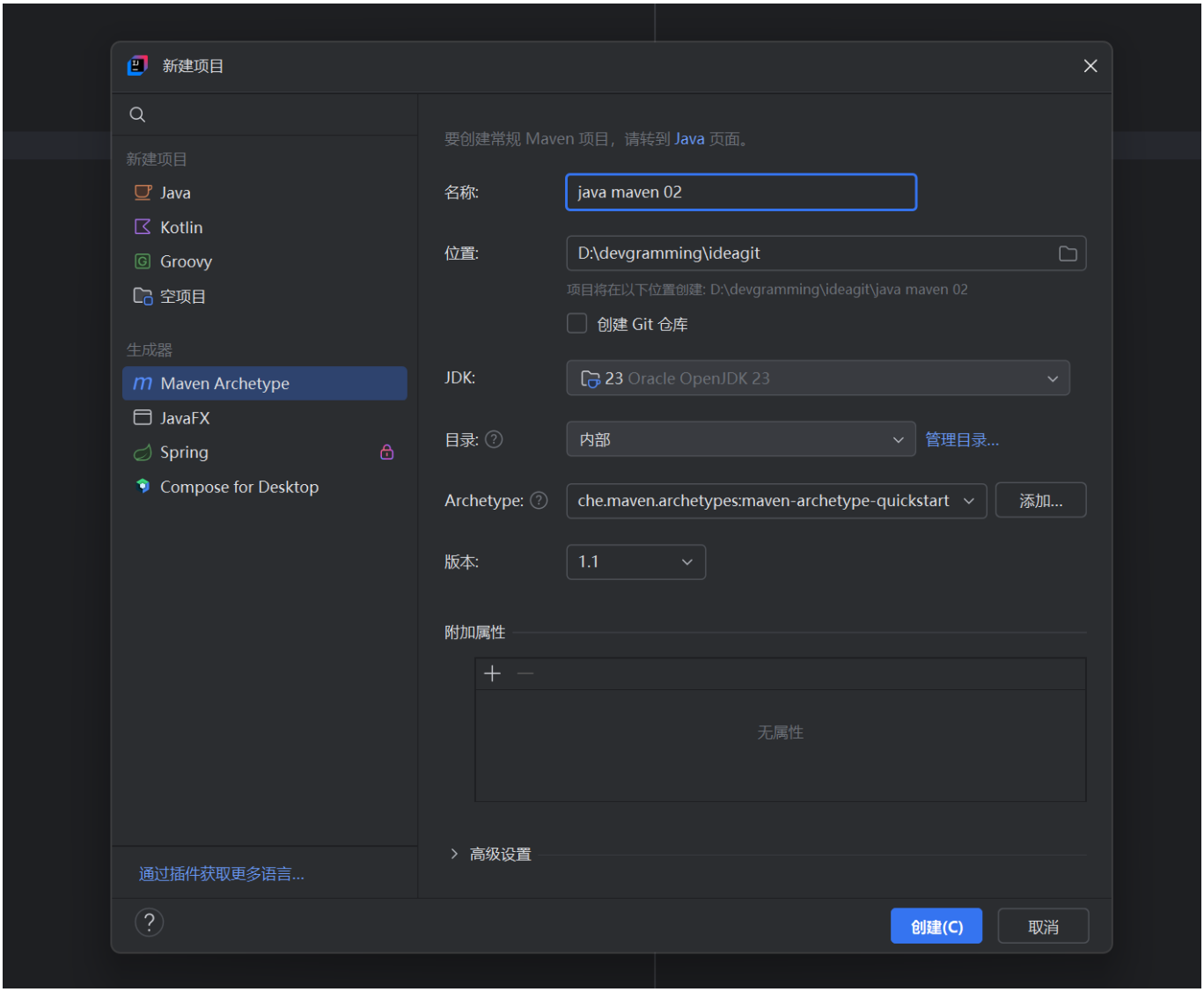
cmd使用mvn -v

```
C:\Users\BEE172>mvn -v
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\devgramming\maven\apache-maven-3.9.9
Java version: 1.8.0_202, vendor: Oracle Corporation, runtime: D:\devgramming\JAVA\JDK\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

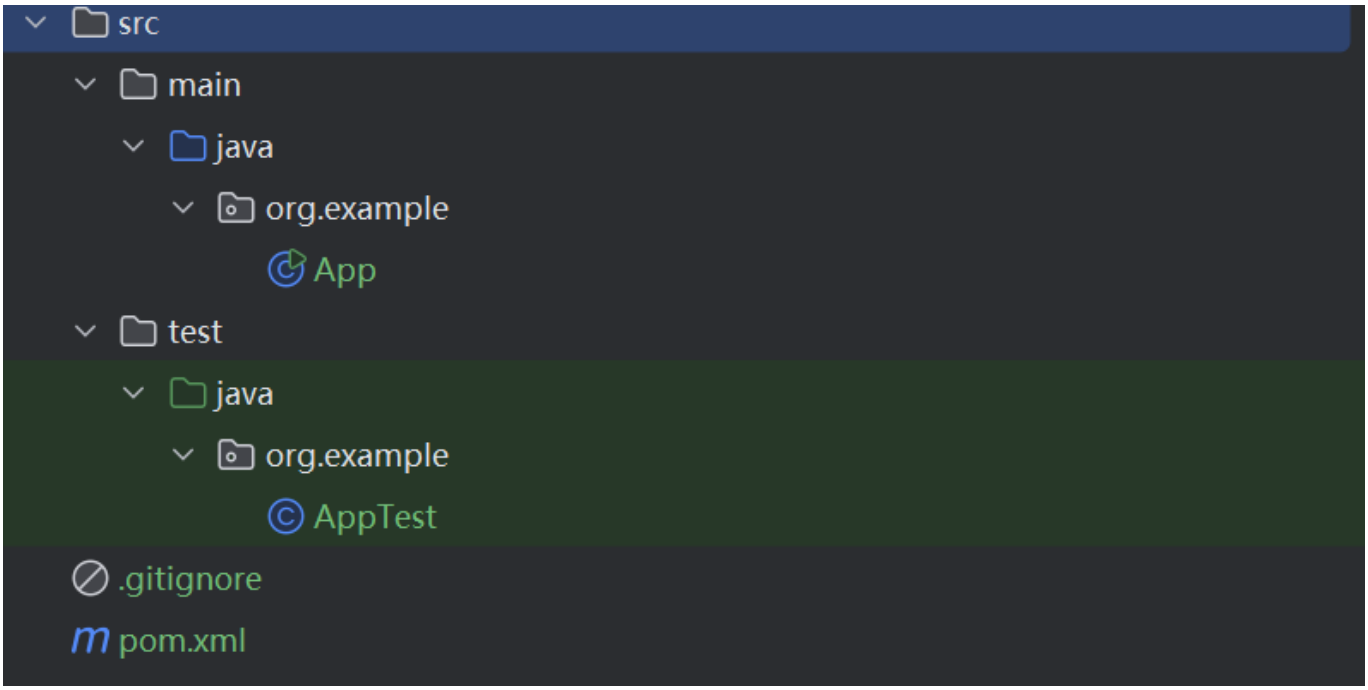
配置环境变量，修改setting文件，挂上镜像仓库后使用cmd，中搭建仓库

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  31.638 s
[INFO] Finished at: 2024-10-02T18:16:57+08:00
[INFO] ------------------------------------------------------------------------
```

idea造项目




# 项目讲解

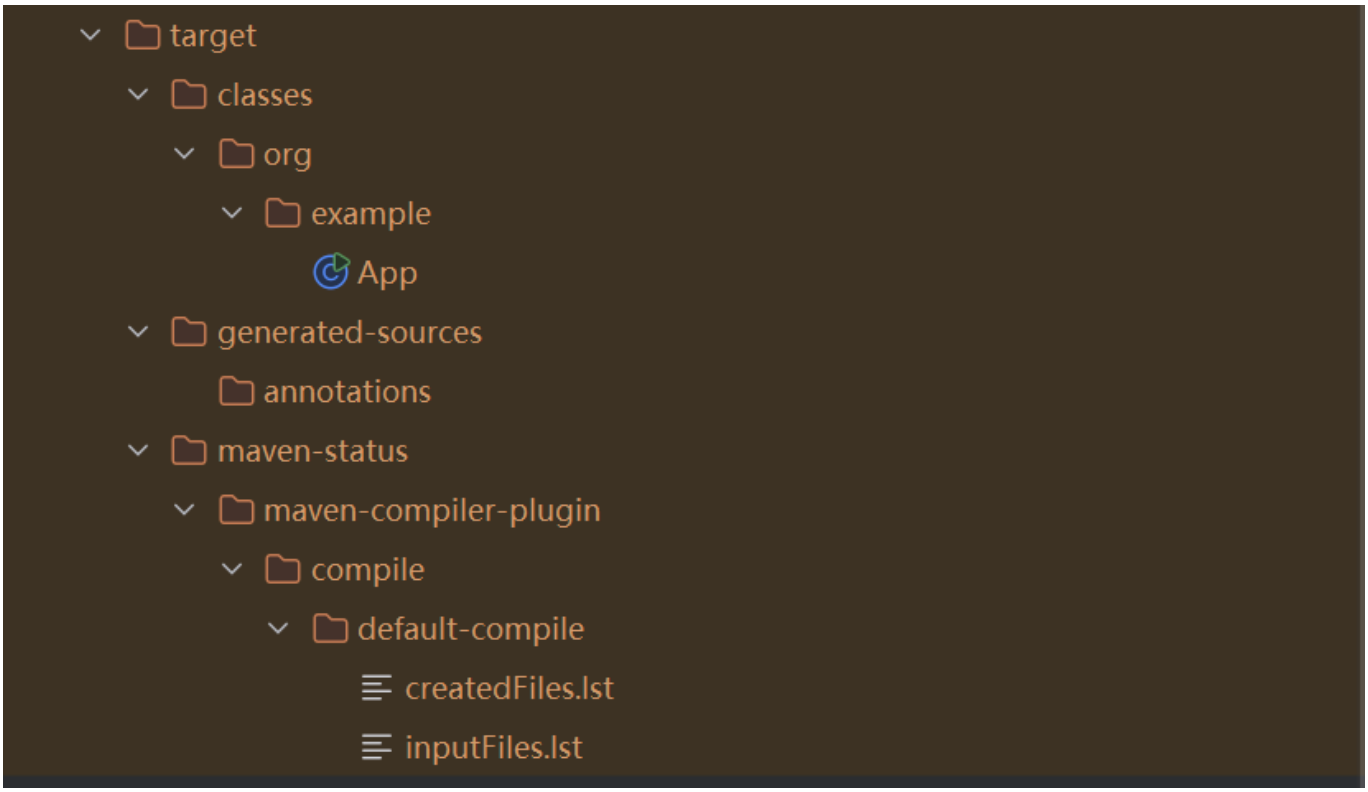



1. pom.xml，项目的核心文件，里面包含项目的配置信息、依赖项、构建指令
2. main/java里存放源代码
3. test/java存放测试代码
4. main中App为quickstart archetype生成的主类，同理test中为测试类

## 体验maven的项目构建

1. clean 清理，在进行真正的构建之前进行一些清理工作，移除所有上一次构建生成的文件。执行该命令会删除项目路径下的target文件，但是不会删除本地的maven仓库已经生成的jar文件。
2. valitate 验证，验证工程是否正确，所需的信息是否完整。
3. compile 编译源码，编译生成class文件,编译命令，只编译选定的目标，不管之前是否已经编译过，会在你的项目路径下生成一个target目录，在该目录中包含一个classes文件夹，里面全是生成的class文件及字节码文件。
4. test 单元测试，测试。
5. package 打包，将工程文件打包为指定的格式，例如JAR，WAR等。这个命令会在你的项目路径下一个target目录，并且拥有compile命令的功能进行编译，同时会在target目录下生成项目的jar/war文件。如果a项目依赖于b项目，打包b项目时，只会打包到b项目下target下，编译a项目时就会报错，因为找不到所依赖的b项目，说明a项目在本地仓库是没有找到它所依赖的b项目，这时就用到install命令了。
6. verify 核实，检查package是否有效、符合标准。
7. install 安装至本地仓库，将包安装至本地仓库，以让其它项目依赖。该命令包含了package命令功能，不但会在项目路径下生成class文件和jar包，同时会在你的本地maven仓库生成jar文件，供其他项目使用（如果没有设置过maven本地仓库，一般在用户/.m2目录下。如果a项目依赖于b项目，那么install b项目时，会在本地仓库同时生成pom文件和jar文件，解决了上面打包package出错的问题）
8. build 功能类似compile，只是只对整个项目进行编译。
9. site 站点，生成项目的站点文档。
10. deploy 复制到远程仓库。 参考https://blog.csdn.net/qq_27022241/article/details/108473343

## 输出目录




1. classes:编译好的文件
2. generated-sourcesMaven 插件在编译过程中生成的源代码
3. createdFiles.lst 列出了编译过程中创建的文件
4. inputFiles.lst 列出了编译过程中的输入文件，即源代码文件

## maven的依赖管理

在pom.xml文件的标签内添加新的依赖项 可填这四个标签 groupId：指定依赖项的groupId，项目的组名 artifactId：指定依赖项的artifactId，项目的唯一标识符 version：指定依赖项的版本号。 scope：指定依赖项在项目中的使用范围

```
<dependency>
    <groupId>name</groupId>
    <artifactId>log4j</artifactId>
    <version>1.7.10</version>
    <scope>test</scope>
</dependency>
```

满足这样就算添加了，这里我只是创建了一个例子