

后端

Java02Plus

4.

那么JVM应该去哪找这个字节码文件呢？**请了解**classpath并给出你对它的理解。如果JVM没找到这个字节码文件会发生什么呢？classpath是一串目录，那么如果两个目录都有一个相同的字节码文件，JVM的选择策略是什么呢？**

JVM会到classpath指定的目录里去寻找这个字节码文件。

classpath是一个环境变量，它会告诉系统我们需要的Java文件储存在哪里，当然它也可以包含很多路径，查找时会——搜索。

那肯定是报错啊，ClassNotFoundException

上面说了，先搜索到谁就用谁，所以要尽量避免这个情况。

5.

一个项目往往有很多class字节码文件，放在不同的package下，也就是不同的目录层次下，如果别人想使用这个项目的某个类，他就需要维护一个完整的目录层次结构，同样的，如果你想把这个项目生成的字节码迁移到生产环境部署运行，也不便于管理，所以将所有的目录层次和字节码文件打成一个文件，就叫做jar包。**请了解*jar包并结合classpath给出你对它的理解，以及回答jar包中**的** /META-INF/MANIFEST.MF 文件是什么，它是必需的吗，有什么作用。**

首先我们要知道的是jar包的定义

JAR (Java Archive) 包是一种将多个类文件、资源文件和元数据打包成一个单独文件的方式。它可以方便地分发和部署 Java 项目，并且可以减少文件数量和管理复杂性。

那么同时jar包也可以被添加到classpath中，所以它本质上就是一个package，一个类似头文件的东西，通常实现有实现一定功能的一些方法。

/META-INF/MANIFEST.MF文件是清单文件，里面有jar包中各种信息，如主类名称，依赖的其它jar包之类。但其实他并不是必须的，但多数情况下它还是很重要的，比如我们可以用它指定主类名称等等。

6.

将Task2的项目手动打包成一个jar包，命名为A.jar（需要编写MAINIFEST文件，不要使用工具，比如maven），然后运行这个jar包（提示：一个最简单的MAINIFEST文件只需要编写Main-Class、Class-Path这两个属性）。

步骤如下：

1. 创建项目，并复制02题代码
2. 创建MANIFEST.MF文件，编辑Main-Class和Class-Path
3. 在根目录执行命令jar cvfm A.jar MANIFEST.MF *

出了些问题大概就是这条命令不给执行

```
PS C:\Users\XYXYXY\Desktop\maventest\A> jar cvfm A.jar MANIFEST.MF *
jar : 无法将“jar”项识别为 cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请确保路径正确，
然后再试一次。
所在位置 行:1 字符: 1
+ jar cvfm A.jar MANIFEST.MF *
+ ~~~
+ CategoryInfo          : ObjectNotFound: (jar:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\XYXYXY\Desktop\maventest\A> |
```

查了资料也没解决，最后用idea导出的一个jar包...

(嗯嗯，已上传，估计会有点难找，github里堆了很多文件了)

7.

在这个项目中，我们只用了自己的类和java的核心类，如果我们还使用了B.jar里的类，就可以把引用的jar包添加到classpath，让我们项目的A.jar包运行起来。但是这样做并没有把B.jar包直接打包进A.jar包，这也是经常发生的情况。**请你了解包的依赖，并说说上面这种情况可能会带来什么问**题，最好举一个具体的例子说明。****

包的依赖，因为附加题中让我们学了Maven怎么使用，肯定也了解了依赖啥的，我这里直接写了：包的依赖是指一个项目需要其它项目里的类。在上述的情况下

我们就必须把A,B包一起部署，并设置一个正确的classpath，后期可能还会出现兼容问题。

例子：在B包的更新中，一个方法名称发生了改变，A在调用B时就会报错，需要对A也进行更新后才能使用。

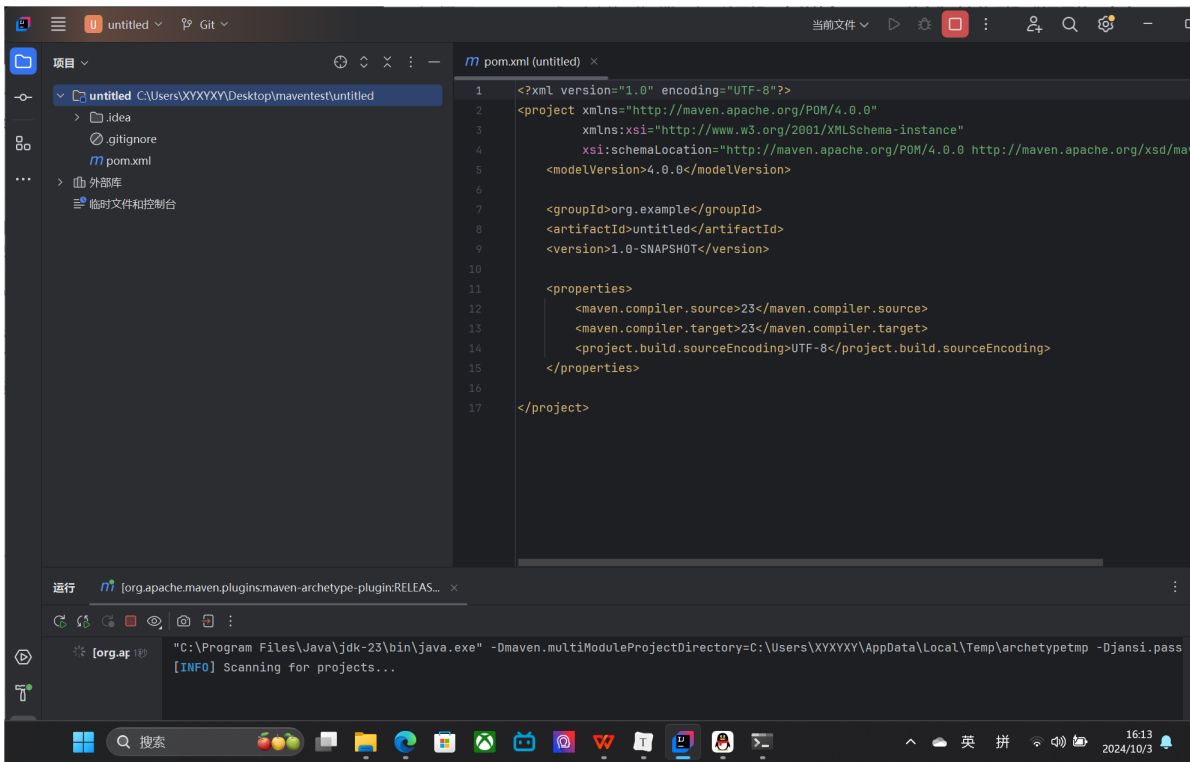
Maven

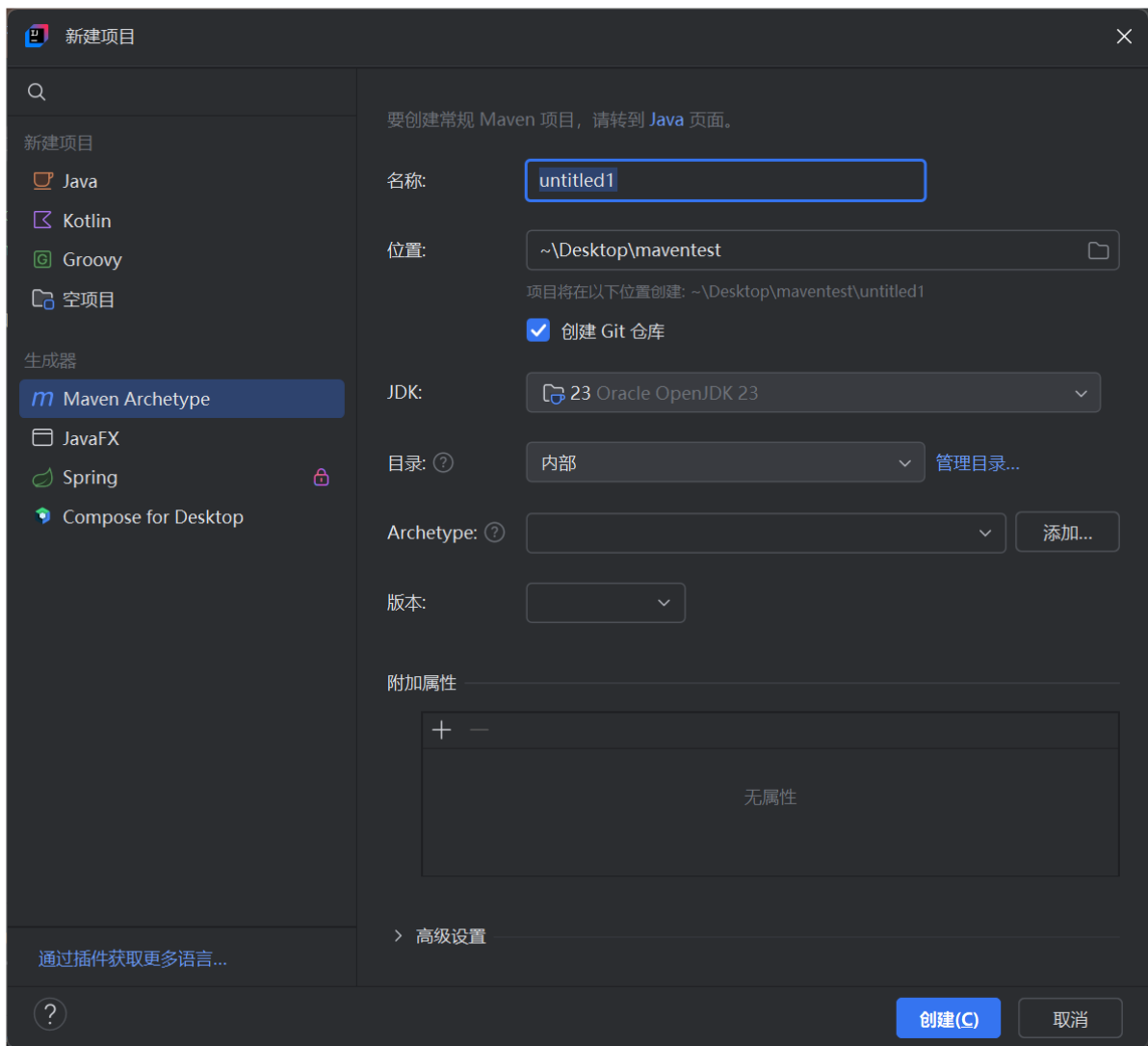
这个在附加题中已经使用过，这里直接放截图了。

镜像配置没有截图，但是搞了的。

安装到IDEA也是。

1. 项目构建






2. 依赖管理

```
















31  @+
32      <dependency>
33          <groupId>org.springframework.boot</groupId>
34          <artifactId>spring-boot-starter-test</artifactId>
35          <scope>test</scope>
36      </dependency>
37  @+
38      <dependency>
39          <groupId>org.springframework.boot</groupId>
40          <artifactId>spring-boot-starter-web</artifactId>
41          <scope>compile</scope>
42      </dependency>
43  @+
44      <dependency>
45          <groupId>org.springframework.boot</groupId>
46          <artifactId>spring-boot-starter-websocket</artifactId>
47      </dependency>
48  @+
49      <dependency>
50          <groupId>mysql</groupId>
51          <artifactId>mysql-connector-java</artifactId>
52          <scope>runtime</scope>
53      </dependency>
54  @+
55      <dependency>
56          <groupId>org.mybatis.spring.boot</groupId>
57          <artifactId>mybatis-spring-boot-starter</artifactId>
58      </dependency>
59  @+
60      <dependency>
61          <groupId>org.projectlombok</groupId>
62          <artifactId>lombok</artifactId>
63      </dependency>
64  @+
65      <dependency>
66          <groupId>com.alibaba</groupId>
67          <artifactId>fastjson</artifactId>
68      </dependency>
69  @+
70      <dependency>

```

>  生命周期

>  插件

✓  依赖项

- >  com.glimmer:monitor-common:1.0-SNAPSHOT
- >  com.glimmer:monitor-pojo:1.0-SNAPSHOT
- >  org.springframework.boot:spring-boot-starter:2.7.3
- >  org.springframework.boot:spring-boot-starter-test:2.7.3 (test)
- >  org.springframework.boot:spring-boot-starter-web:2.7.3
- >  org.springframework.boot:spring-boot-starter-websocket:2.7.3
-  mysql:mysql-connector-java:8.0.30 (runtime)
- >  org.mybatis.spring.boot:mybatis-spring-boot-starter:2.2.0
-  org.projectlombok:lombok:1.18.20
-  com.alibaba:fastjson:1.2.76
- >  com.alibaba:druid-spring-boot-starter:1.2.1
- >  com.github.pagehelper:pagehelper-spring-boot-starter:1.3.0
-  org.aspectj:aspectjrt:1.9.4
-  org.aspectj:aspectjweaver:1.9.4
- >  javax.xml.bind:jaxb-api:2.3.1

笔记part:

\1. 生命周期的功能:

\1. clean: **清理项目**, 将编译生成的字节码文件和jar包文件删除

\2. validate: **验证**项目是否正确, 并且所有必要的信息是可用的

\3. compile: **编译**项目源代码, 生成字节码文件

\4. test: **单元测试**, 会执行我们test目录下的test用例

\5. package: **打包**项目, 把编译生成的字节码文件和其他的资源文件一起打包生成jar包或是war包 (其实在执行package时也会先编译和测试一下, 没问题后才会打包)

\6. verify: **检查**打包生成的jar包是否正确

\7. install: 把打包生成的jar包或是war包**安装**到本地仓库

\8. deploy: 把打包好的jar包**上传**到远程仓库里

\9. site: **生成**项目站点文档

\2. 依赖管理:

\1. provided: 编译时需要, 运行时不需要

\2. test: 依赖只在测试时需要 (不会被打包到jar包中)

\3. compile: 编译和运行时都需要

\4. runtime: 运行时需要, 编译时并不需要
5. system: 本地提供的依赖, 此时还需要一个systempath来指定该去哪里找到这个依赖 (不过

会导致可移植性变差, 最好还是上传私服)

\6. import: 导入其它pom文件里的依赖, 但不会实际引入依赖

\3. 依赖添加: 搜索然后添加: 在官网找到下载路径, 复制到pom的dependency中就可以了 (第一次下会爆红,

刷新一下, 它会帮你下载下来)

\4. 依赖传递: 好像不是一个我需要关心的问题, 别人已经帮你搞好了 (只有是compile的依赖会被传递)

\5. 依赖冲突: 如果我们的两个依赖分别依赖了一个依赖的不同版本, 这个时候idea会选择最短路径优先, 然后是先声明优先。

当然也可以手动控制依赖, 用exclusions标签来排除不需要的依赖, 用optional标签来标记一个依赖是可选的

(这部分是在附加题中直接拷贝的。)