Java10

Task1.maven

1. 什么是maven?

1.1 什么是maven

Maven是一个**项目管理与构建自动化工具**，主要用于Java项目，但也可用于其他语言（如Kotlin、Scala）。

Maven 解决了**软件构建**的两方面问题：一是软件是如何构建的。二是软件的依赖关系。

Maven 的**核心功能**包括：**项目构建**（编译、测试、打包、部署），**依赖管理**（自动下载和管理第三方库），**标准化项目结构**（约定优于配置），**插件扩展**（支持自定义构建流程）。

Maven的主要功能和优势：

Maven 的主要功能和优势

功能	说明
依赖管理	自动下载和管理 `.jar` 文件，避免手动管理依赖
标准化构建流程	提供 `clean`、`compile`、`test`、`package` 等标准生命周期
项目模板（Archetype）	快速生成项目结构（如 `maven-archetype-quickstart`）
多模块支持	适用于大型项目，可以拆分为多个子模块
插件扩展	支持自定义构建任务（如 `maven-compiler-plugin` 指定 Java 版本）

优势：

- **减少配置**：约定优于配置，减少 `build.xml`（Ant）这样的手动配置。
- **依赖自动管理**：只需声明依赖，Maven 自动下载并处理冲突。
- **跨平台**：基于 Java，可在 Windows、Linux、macOS 上运行。
- **与 IDE 集成**：Eclipse、IntelliJ IDEA、VS Code 都支持 Maven。

> 来自https://www.runoob.com/maven/maven-intro.html

1.2 什么是jar包

JAR (Java ARchive) 是一种 Java 打包文件格式，它可以 ***将一组 Java 类文件、相关资源和元数据打包到一个单独的文件中***。JAR 文件通常用于分发 Java 应用程序或库，并且可以在不同平台上运行。

JAR 包是一种方便的方式来组织和管理 Java 代码和依赖项，并且可以轻松地与其他开发人员共享和部署代码。JAR 包还支持数字签名和版本控制，以确保代码的完整性和安全性。

> 来自csdn的一篇文章

1.3 maven和jar包有关系吗？有什么关系?

有关系。

Maven和JAR包的关系主要体现在以下几个方面：

- **依赖管理**：Maven用于自动管理Java项目中的JAR包依赖，避免手动导入JAR包的繁琐过程。
- **冲突解决**：Maven能够防止不同JAR包之间的依赖冲突，确保项目的稳定性。
- **传递依赖**：Maven支持传递依赖特性，即如果项目A依赖于B，而B又依赖于C，Maven会自动引入C，无需在pom.xml中显式声明。
- **构建过程**：Maven可以通过插件将项目打包成JAR文件，并管理构建生命周期。

通过这些功能，Maven极大地简化了Java项目的依赖管理和构建过程。

1.4 我们为什么要用maven?

Maven 通过标准化和自动化，极大地提升了 Java 项目的可维护性、开发效率和工程化水平。

总结：我们为什么要用 Maven?

痛点	Maven 的解决方案	带来的好处
手动管理 JAR 包，依赖混乱	**自动依赖管理**	提升开发效率，杜绝依赖冲突
项目结构五花八门	**标准化的项目结构**	降低学习成本，便于团队协作
构建过程复杂且不统一	**统一的构建生命周期和命令**	构建过程自动化、标准化、可重复
项目信息分散	**集中化的项目对象模型**	便于项目管理和维护
多模块项目难以管理	**项目继承和聚合**	支持复杂的企业级项目架构

> 来自deepseek

2. 试着下载maven吧!

> 参考b站教程

从https://maven.apache.org/download.cgi上下载了：

Downloading Apache Maven 3.9.11

Apache Maven 3.9.11 is the latest release: it is the recommended version for all users.

System Requirements

Java Development Kit (JDK)	Maven 3.9+ requires JDK 8 or above to execute. It still allows you to build against 1.3 and other JDK versions by using toolchains.
Memory	No minimum requirement
Disk	Approximately 10MB is required for the Maven installation itself. In addition to that, disk space will be used for your local Maven repository. The size of your local repository will vary depending on usage but expect at least 500MB.
Operating System	No minimum requirement. Start up scripts are included as shell scripts (tested on many Unix flavors) and Windows batch files.

Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

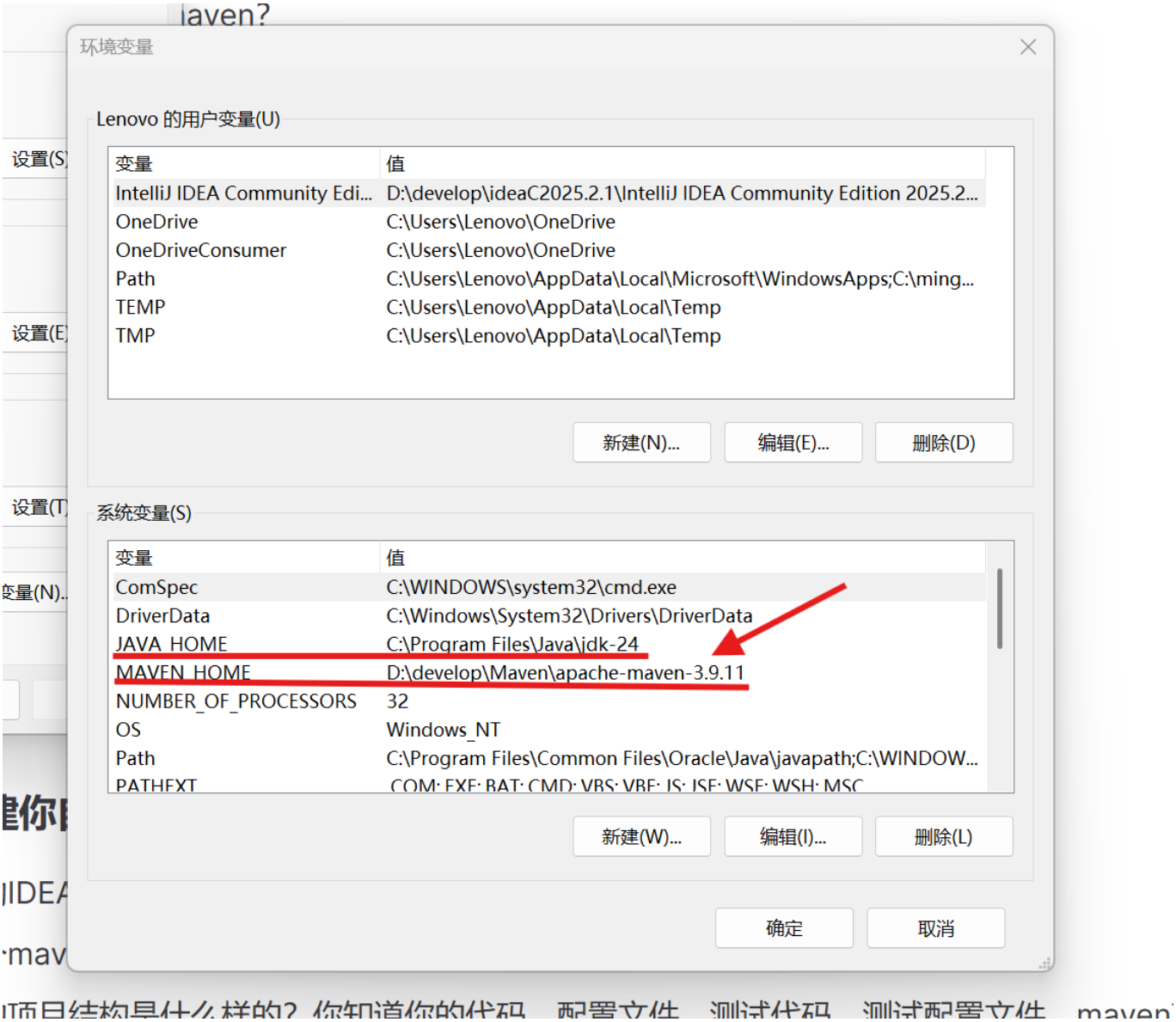
In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.9.11-bin.tar.gz	apache-maven-3.9.11-bin.tar.gz.sha512	apache-maven-3.9.11-bin.tar.gz.asc
Binary zip archive	apache-maven-3.9.11-bin.zip	apache-maven-3.9.11-bin.zip.sha512	apache-maven-3.9.11-bin.zip.asc
Source tar.gz archive	apache-maven-3.9.11-src.tar.gz	apache-maven-3.9.11-src.tar.gz.sha512	apache-maven-3.9.11-src.tar.gz.asc
Source zip archive	apache-maven-3.9.11-src.zip	apache-maven-3.9.11-src.zip.sha512	apache-maven-3.9.11-src.zip.asc

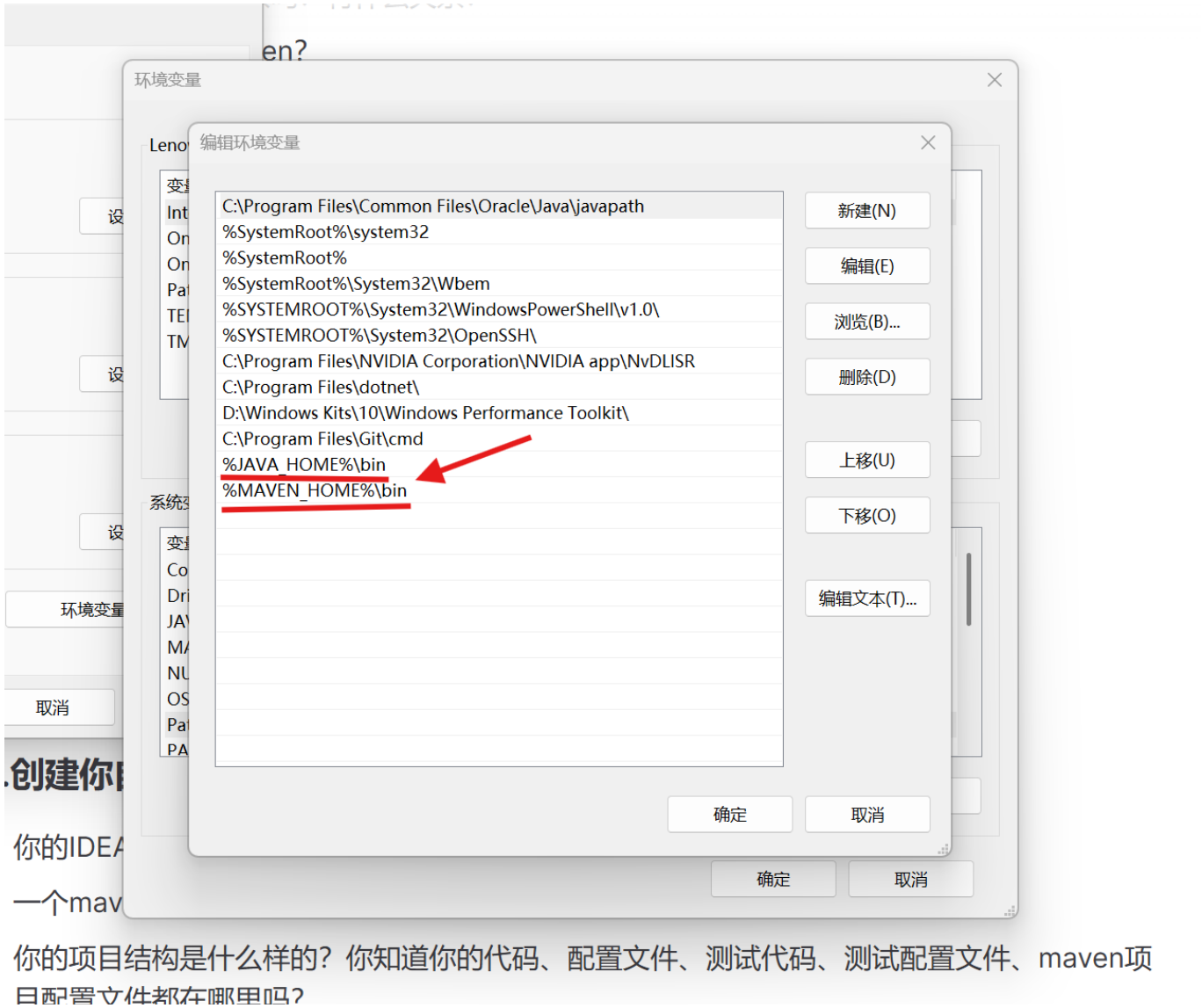
- 3.9.11 Release Notes and Release Reference Documentation
- latest source code from source repository
- Distributed under the Apache License, version 2.0
- other:
  - All current release sources (plugins, shared libraries,...) available at https://downloads.apache.org/maven/

环境配置：

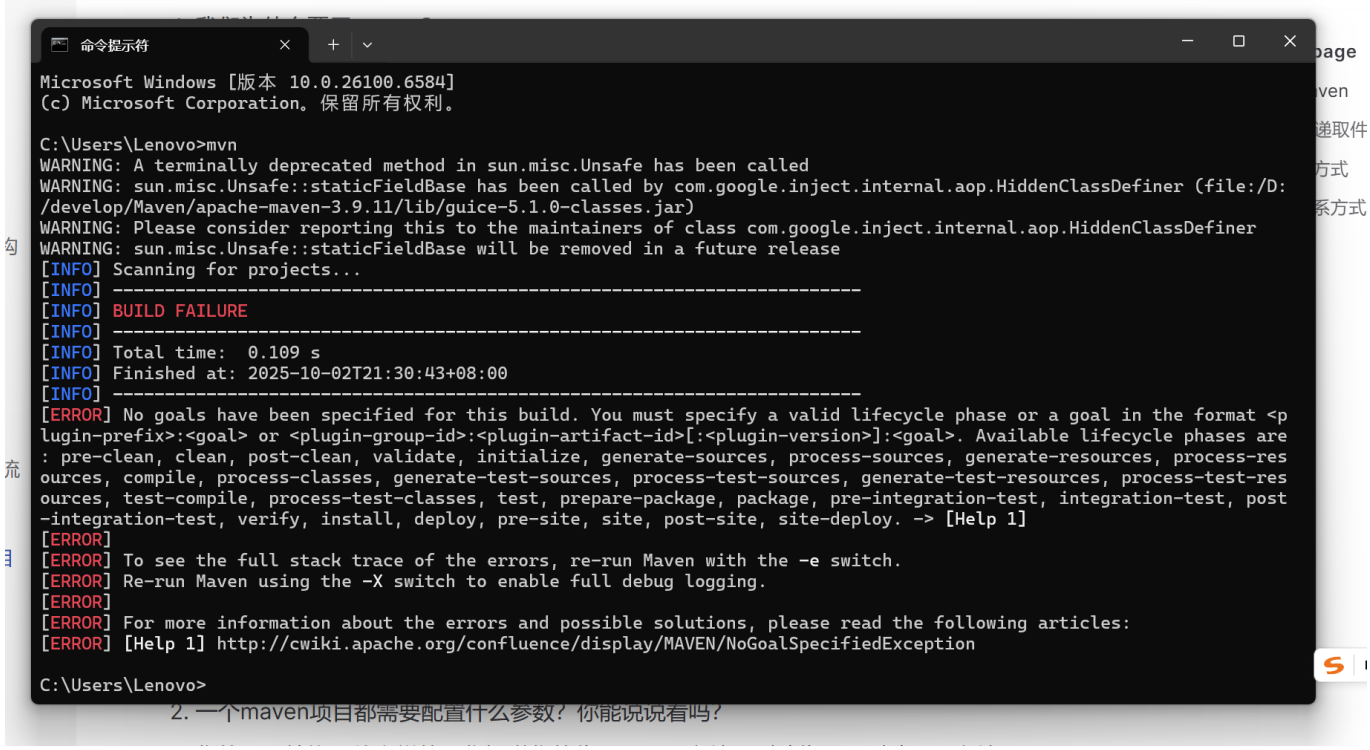
系统变量：


环境变量
Lenovo 的用户变量(U)
变量 值
IntelliJ IDEA Community Edi... D:\develop\ideaC2025.2.1\IntelliJ IDEA Community Edition 2025.2...
OneDrive C:\Users\Lenovo\OneDrive
OneDriveConsumer C:\Users\Lenovo\OneDrive
Path C:\Users\Lenovo\AppData\Local\Microsoft\WindowsApps;C:\ming...
TEMP C:\Users\Lenovo\AppData\Local\Temp
TMP C:\Users\Lenovo\AppData\Local\Temp
新建(N)... 编辑(E)... 删除(D)
系统变量(S)
变量 值
ComSpec C:\WINDOWS\system32\cmd.exe
DriverData C:\Windows\System32\Drivers\DriverData
JAVA_HOME C:\Program Files\Java\jdk-24
MAVEN_HOME D:\develop\Maven\apache-maven-3.9.11
NUMBER_OF_PROCESSORS 32
OS Windows_NT
Path C:\Program Files\Common Files\Oracle\Java\javapath;C:\WINDOW...
新建(W)... 编辑(I)... 删除(L)
确定 取消

Path：



cmd测试环境配置结果:



版本信息:

```
命令提示符
Microsoft Windows [版本 10.0.26100.6584]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Lenovo> mvn -v
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
Maven home: D:\develop\Maven\apache-maven-3.9.11
Java version: 24.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-24
Default locale: zh_CN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Lenovo>
```

3.什么是maven仓库?

> 参考b站教程

仓库配置

配置本地仓库位置:

settings.xml

D: > develop > Maven > apache-maven-3.9.11 > conf > settings.xml

```
<!--
  |                    ${maven.conf}/settings.xml.
  |
  |                    NOTE: This location can be overridden with the CLI option:
  |
  |                    -gs /path/to/global/settings.xml
  |
  | The sections in this sample file are intended to give you a running start at
  | getting the most out of your Maven installation. Where appropriate, the default
  | values (values used when the setting is not specified) are provided.
  |
  |-->
<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">
  <!-- localRepository
   | The path to the local repository maven will use to store artifacts.
   |
   | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  -->
  <localRepository>D:\develop\Maven\repository</localRepository>
  <!-- interactiveMode
   | This will determine whether maven prompts you when it needs input. If set to false,
   | maven will use a sensible default value, perhaps based on some other setting, for
   | the parameter in question.
   |
   | Default: true
  <interactiveMode>true</interactiveMode>
  -->

  <!-- offline
   | Determines whether maven should attempt to connect to the network when executing a build
```

配置镜像仓库：

settings.xml

D: > develop > Maven > apache-maven-3.9.11 > conf > settings.xml

```
46   <settings xmlns="http://maven.apache.org/SETTINGS/1.2.0"
147    <mirrors>
148      <!-- mirror
152      |
153      <mirror>
154        <id>mirrorId</id>
155        <mirrorOf>repositoryId</mirrorOf>
156        <name>Human Readable Name for this Mirror.</name>
157        <url>http://my.repository.com/repo/path</url>
158      </mirror>
159       -->
160      <mirror>
161        <id>maven-default-http-blocker</id>
162        <mirrorOf>external:http:*</mirrorOf>
163        <name>Pseudo repository to mirror external repositories initially using HTTP.</name>
164        <url>http://0.0.0.0/</url>
165        <blocked>true</blocked>
166      </mirror>
167      <mirror>
168        <id>aliyun-maven</id>
169        <mirrorOf>central</mirrorOf>
170        <url>https://maven.aliyun.com/repository/public</url>
171        <blocked>false</blocked>
172      </mirror>
173    </mirrors>
174
175    <!-- profiles
```

中央仓库和私服分别是什么

中央仓库是由 Maven 社区维护的、默认的、全球唯一的仓库。它有如下特点：

- **公开性**：对所有开发者开放，任何人都可以从这里下载依赖。
- **权威性**：它是 Maven 世界的标准源，包含了绝大多数流行的开源 Java 库（如 Spring、Jackson、Logback 等）的官方版本。
- **无需配置**：Maven 默认就知道它的地址，你不需要在任何配置文件中声明它。
- **只读性**：普通开发者无法将自己项目的构件上传到中央仓库。

当项目里声明了一个依赖，Maven 首先会检查本地仓库，如果没有找到，它就会自动前往中央仓库下载。

但它也有局限性：

1. **网络依赖**：必须能访问互联网。
2. **国内速度慢**：服务器在国外，国内下载可能较慢。
3. **只有开源的资源**：公司内部开发的、不开源的 jar 包，不可能出现在这里。

私服是在局域网内搭建的一个私有仓库服务器，它代理并缓存了外部公共仓库（如中央仓库）。它有如下特点：

- **私有性**：属于某个组织，通常部署在内网。
- **代理和缓存**：当有开发者请求一个依赖时，私服会代替开发者去外部的中央仓库或其他公共仓库下载，并缓存在本地。下一个开发者再请求同样的依赖时，就直接从私服的高速内网获取，速度极快。

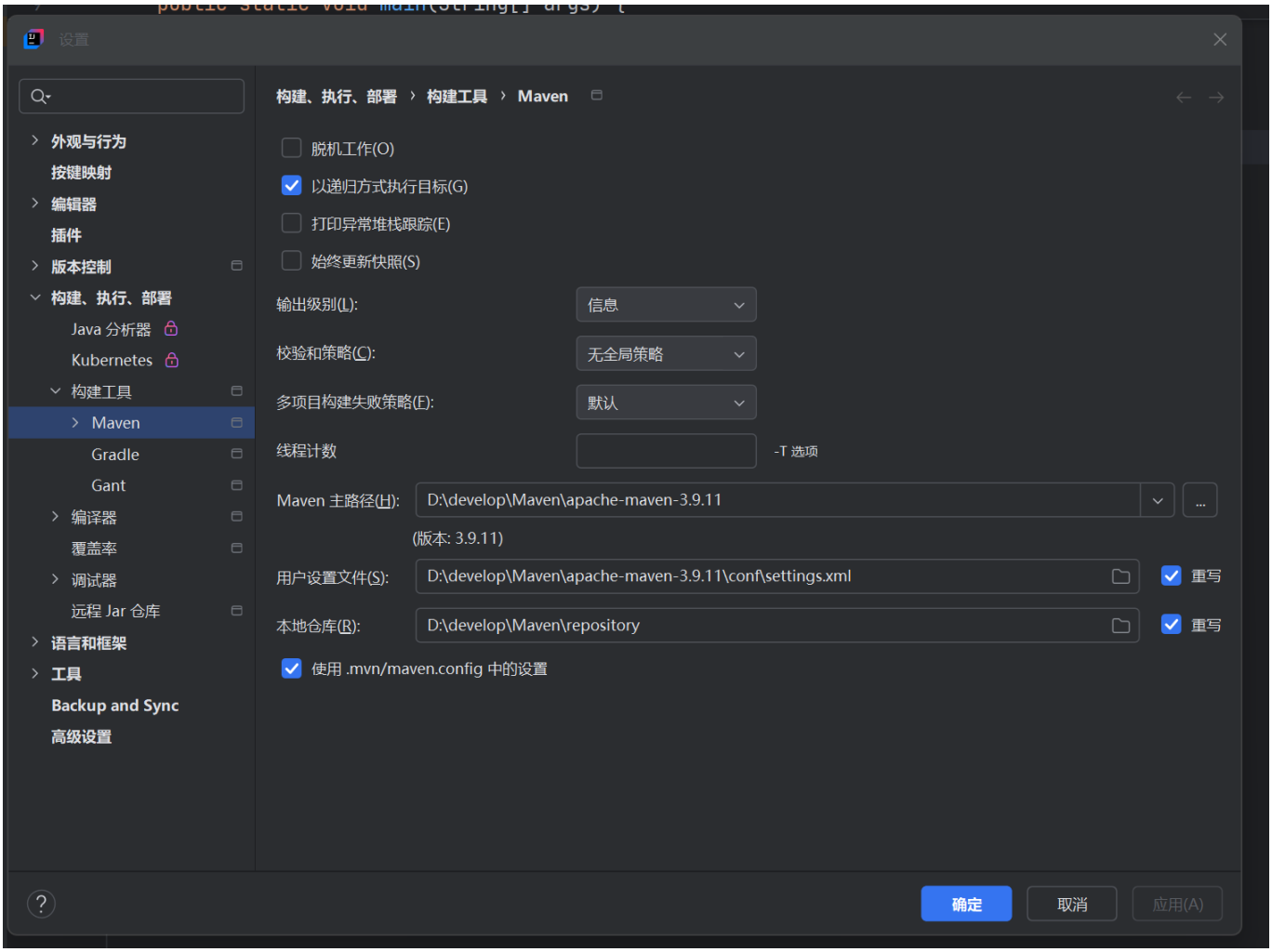
它可以：

1. **加速构建**：内网传输，速度快；同时缓存避免了重复下载。
2. **隔离外网**：即使外网中央仓库不可用，只要私服上有缓存，内网开发构建就不会中断。这提供了稳定性。
3. **共享**：作为内部模块的共享中心。
4. **安全与控制**：可以管理哪些依赖能被下载，防止引入不安全的组件。

4. 创建你自己的maven项目！

4.1 IDEA关联maven

> 参考b站视频



4.2 maven项目需要配置的参数

> 参考deepseek

一个 Maven 项目的配置可以大致分为两类：**项目核心配置**（在 pom.xml 中）和**环境运行配置**（在 settings.xml 中或命令行参数）。

项目核心配置 (pom.xml):

1. 项目坐标（`groupId`、`artifactId`等）：这是项目的唯一标识。
2. 父项目继承（`parent`）：如果项目继承自一个父 POM（比如 Spring Boot），可以统一管理依赖版本。

3. 属性定义（`properties`）：用于定义常量，便于统一管理和引用。
4. 依赖管理（`dependencies`）：声明项目所依赖的外部库。
5. 构建配置（`build`）：配置如何编译、打包项目。
6. 仓库配置（`repositories`）：如果需要使用非中央仓库的第三方库。
7. 多模块配置（`modules`）：如果是聚合项目，需要声明包含哪些子模块。

环境运行配置 (settings.xml)：

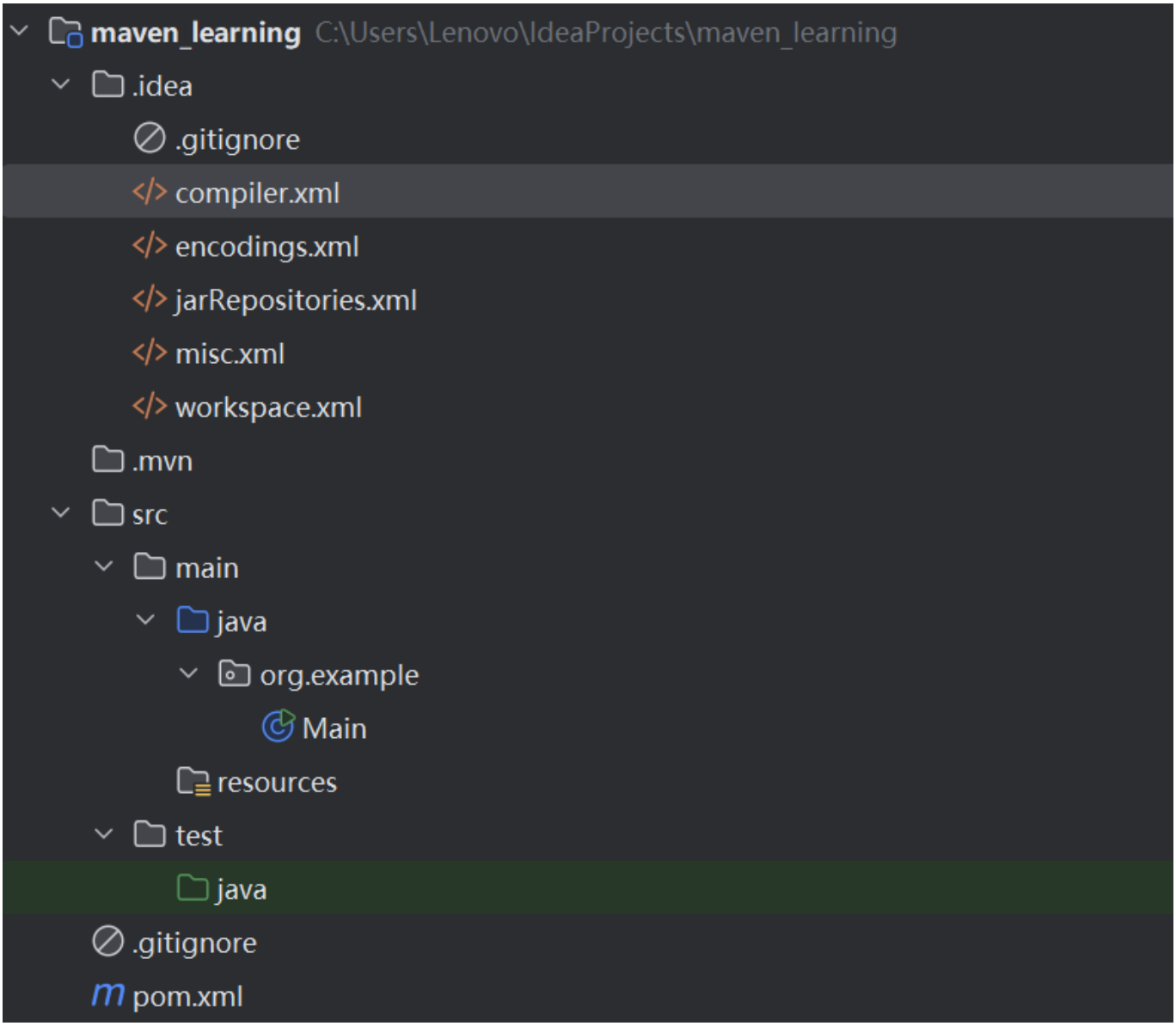
1. 本地仓库路径（`localRepository`）
2. 镜像仓库（`mirrors`）：最重要、最常用的配置之一，用于加速下载或连接私服。
3. 服务器认证信息（`servers`）：访问私有仓库时需要用户名和密码。
4. 代理配置（`proxies`）：如果公司网络需要代理才能访问外网。
5. Profiles（`profiles`）：用于不同环境的配置切换（如开发、测试、生产）。

命令行参数：

- 指定 Profile：`mvn clean install -P dev,prod`
- 跳过测试：`mvn clean install -DskipTests`
- 设置系统属性：`mvn test -Dapp.env=test -Ddb.port=3307`

4.3 项目结构

使用IDEA新建的项目的结构：



其中，

maven_learning/目录是项目根目录。

.mvn/目录包含 Maven Wrapper 相关文件。它确保即使开发者的机器上没有安装 Maven，或者安装的版本不一致，项目也能用正确的 Maven 版本构建。可以使用 `./mvnw`（Linux/Mac）或 `mvnw.cmd`（Windows）代替 `mvn` 命令。

src/目录存放源代码，`src/main/java`中存放各种类的java源代码，`src/main/resources`中存放存非代码的配置文件（如Spring Boot 等框架的配置文件）和静态资源（如CSS, JS, 图片等 web 静态资源），`src/test/java`中存放测试代码（包结构应与 main/java 保持一致）。

pom.xml文件是项目对象模型 （核心配置文件），存放该项目的各种基本信息。

.gitignore文件用于告诉 Git 哪些文件或目录不应该纳入版本控制。

.idea/目录包含 IntelliJ IDEA 的特定配置，这些配置通常不应该提交到版本库（这也是为什么里面有 .gitignore 文件）。其中包括：**compiler.xml**编译器设置、**encodings.xml**文件编码设置、**jarRepositories.xml**仓库配置、**misc.xml**杂项配置、**workspace.xml**工作空间设置（窗口布局、运行配置等）。

此外，

还有一个**target/目录**，它位于项目根目录下，在执行 `mvn compile` 或 `mvn package` 后自动生成。它包含所有构建产物：编译后的 .class 文件、生成的 JAR 文件、测试报告等。执行 `mvn clean` 会删除此目录。

4.4 pom.xml文件

它是什么?

POM 的全称是 **Project Object Model**（项目对象模型）。它使用 XML 格式来描述项目，定义了**项目的全部信息**，包括：它是什么？（**身份**）它依赖什么？（**第三方库**）它能做什么？（**如何构建和打包**）谁创建的？（**项目信息**）

总结：pom.xml 是 Maven 项目的核心配置文件。

它有什么用?

pom.xml 的主要作用可以归结为以下几点：

1. **项目标识**：通过"坐标"唯一确定一个项目。
2. **依赖管理**：声明项目所依赖的外部库（JAR 包），Maven 会自动从仓库下载并管理它们。
3. **构建生命周期管理**：定义如何编译、测试、打包、部署项目。
4. **项目信息传递**：包含项目描述、开发者信息、许可证等，便于交流和分发。
5. **继承与聚合**：支持多模块项目的管理，实现配置的复用。

它的基本配置有哪些?

一个最基础的 pom.xml 文件包含以下核心部分：

```
<?xml version="1.0" encoding="UTF-8"?>
<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，固定为 4.0.0 -->
    <modelVersion>4.0.0</modelVersion>

    <!-- ### 项目坐标（唯一标识） - 必须 ### -->
    <groupId>org.example</groupId>     <!-- 公司/组织域名反写 -->
    <artifactId>maven_learning</artifactId>      <!-- 项目名 -->
    <version>1.0-SNAPSHOT</version>    <!-- 版本号 -->
    <packaging>jar</packaging>           <!-- 项目类型，默认为 jar -->

    <!-- ### 项目属性 - 可选 ### -->
    <properties>
        <maven.compiler.source>24</maven.compiler.source> <!-- Java 版本 -->
        <maven.compiler.target>24</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> <!-- 编
码 -->
    </properties>

    <!-- ### 依赖管理 - 核心 ### -->
```

```
    <dependencies>
        <!-- 具体依赖在这里声明 -->
    </dependencies>

    <!-- ### 构建配置 - 可选 ### -->
    <build>
        <plugins>
            <!-- 构建插件在这里配置 -->
        </plugins>
    </build>
</project>
```

如何通过这个文件管理jar包版本、项目类型?

> 示例来自deepseek

依赖管理主要在 `<dependencies>` 部分进行。主要有两种方式：

a) 声明一个依赖：

```
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version> <!-- 直接指定版本 -->
        <scope>test</scope> <!-- 指定依赖作用域 -->
    </dependency>
```

b) 统一管理版本：

对于多个相关依赖，可以使用 `<dependencyManagement>` 或 `<properties>` 统一管理版本，避免版本冲突。

方法一：使用 Properties

例如：

```
<properties>
    <spring-boot.version>2.7.0</spring-boot.version>
    <junit.version>4.13.2</junit.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring-boot.version}</version> <!-- 使用属性 -->
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
```

```
    </dependency>
</dependencies>
```

方法二：继承父 POM

很多框架（如 Spring Boot）提供了父 POM 来统一管理所有依赖的版本。例如：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.0</version>
</parent>

<dependencies>
    <!-- 在父POM中已经管理了版本，这里无需再写version -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

项目类型通过 `<packaging>` 标签来定义。它决定了 Maven 使用不同的生命周期和打包方式。

常见的项目类型：

打包类型	说明	产出物	适用场景
`jar`	**Java 应用程序**	`.jar` 文件	普通的 Java 库或可执行应用（默认类型）
`war`	**Web 应用程序**	`.war` 文件	需要部署到 Servlet 容器（如 Tomcat）的 Web 项目
`pom`	**聚合/父项目**	无实际产出	多模块项目的父 POM，仅用于管理子模块和统一配置
`ear`	企业级应用程序	`.ear` 文件	J2EE 企业级应用
`maven-plugin`	Maven 插件	`.jar` 文件	自定义 Maven 插件

示例：

JAR 项目：

```
<packaging>jar</packaging> <!-- 显式声明，但jar是默认值，可省略 -->
```

Web 项目

```
<packaging>war</packaging> <!-- 声明为 WAR 包 -->
```

多模块父项目：

```
<packaging>pom</packaging> <!-- 声明为 POM 类型 -->

<modules>
    <module>core-module</module> <!-- 声明子模块 -->
    <module>web-module</module>
</modules>
```

4.5 如何导入需要的jar包

> 对于中央仓库中存在的依赖，只需要在 pom.xml 文件中添加依赖配置，之后执行 mvn 指令时就会自动将对应依赖下载到本地仓库中（如果本地仓库中没有对应依赖的话）。

方法一：手动在 pom.xml 中添加依赖

1. 打开 pom.xml 文件
2. 在 `<dependencies>` 标签内添加依赖配置
3. 保存文件，IDEA 会自动下载依赖

方法二：从MVNREPOSITORY网站复制

1. 访问 https://mvnrepository.com/
2. 搜索需要的库
3. 选择正确的库和版本
4. 复制 Maven 依赖配置
5. 粘贴到 pom.xml 中

方法三：添加本地 JAR 包 将 JAR 文件安装到本地仓库，然后在 pom.xml 中正常引用。

5. 启动你的maven！

5.1 maven的常用命令

一、项目生命周期核心命令

- `mvn compile`：编译项目的主源代码。编译后的 .class 文件位于 target/classes 目录。
- `mvn test`：运行项目的测试用例（使用 JUnit 或 TestNG）。它会自动先执行 compile 阶段。测试报告通常在 target/surefire-reports 目录。
- `mvn package`：将编译后的代码打包成可分发格式，如 JAR、WAR。它会自动先执行 compile 和 test。生成的包在 target/ 目录下。
- `mvn install`：将项目打包并安装到本地仓库。这样，本地其他项目就可以将它作为依赖来引用了。它会自动执行 compile, test, package。
- `mvn deploy`：将最终的项目包复制到远程仓库。
- `mvn clean`：清理项目，删除 target 目录。这是一个独立的生命周期，用于清除之前构建生成的所有文件，确保下一次构建是“干净的”。
- `mvn clean package`：先清理再打包。这是组合命令，确保打包基于最新代码。

- `mvn clean install`：先清理再安装到本地仓库。这是组合命令。

二、项目创建与信息查询命令

- `mvn archetype:generate`：使用 Maven 原型（模板）快速创建一个新项目骨架。常用参数：`-DgroupId=... -DartifactId=... -DarchetypeArtifactId=... -DinteractiveMode=false`（写在这条命令后面）。
- `mvn dependency:tree`：以树形结构显示项目的所有依赖（包括传递性依赖）。常用于解决 Jar 包冲突时。

> **传递性依赖** 是指：不需要直接声明某个 JAR 包，但项目却能自动引入它。这是因为直接声明的依赖本身也依赖了其他的 JAR 包，Maven 会自动将这些间接的依赖也引入到项目中。

- `mvn dependency:analyze`：分析项目的依赖，检查哪些依赖声明了但未使用，哪些使用了但未声明。
- `mvn help:effective-pom`：显示项目的有效 POM。它会合并当前项目的 POM、父 POM 以及 Super POM 的所有配置。

> **Super POM** 是 Maven 的默认父 POM。所有的 Maven 项目（无论是否显式声明了父 POM）都隐式地继承自 Super POM。（类似 Java 中的 Object 类）

- `mvn help:describe`：描述一个插件或目标的属性。

三、与 IDE 集成和跳过测试的命令

- `mvn eclipse:eclipse`：生成 Eclipse 项目文件（.project, .classpath）。

> **Eclipse** 是一个开源的、跨平台的集成开发环境。

- `mvn idea:idea`：生成 IntelliJ IDEA 项目文件。
- `mvn clean install -DskipTests`：编译测试代码，但在运行阶段跳过测试。
- `mvn clean install -Dmaven.test.skip=true`：完全跳过测试代码的编译和执行。（更彻底的跳过）

> `-DskipTests` 参数可以加在任何生命周期命令后面，如 `mvn package -DskipTests`。

四、高级和实用命令

- `mvn versions:set -DnewVersion=1.1.0`：批量更新项目的版本号（包括子模块）。
- `mvn source:jar`：将项目源代码打包成一个 JAR 文件。
- `mvn site`：生成项目相关的站点文档。

> **站点文档**是一个专门为项目自动生成的网站，它旨在提供关于项目的、超越代码本身的综合性技术信息。运行 mvn site 后，Maven 会在 target/site/ 目录下生成一系列的 HTML、CSS 和图片文件,可以直接用浏览器打开 target/site/index.html 来查看这个网站。

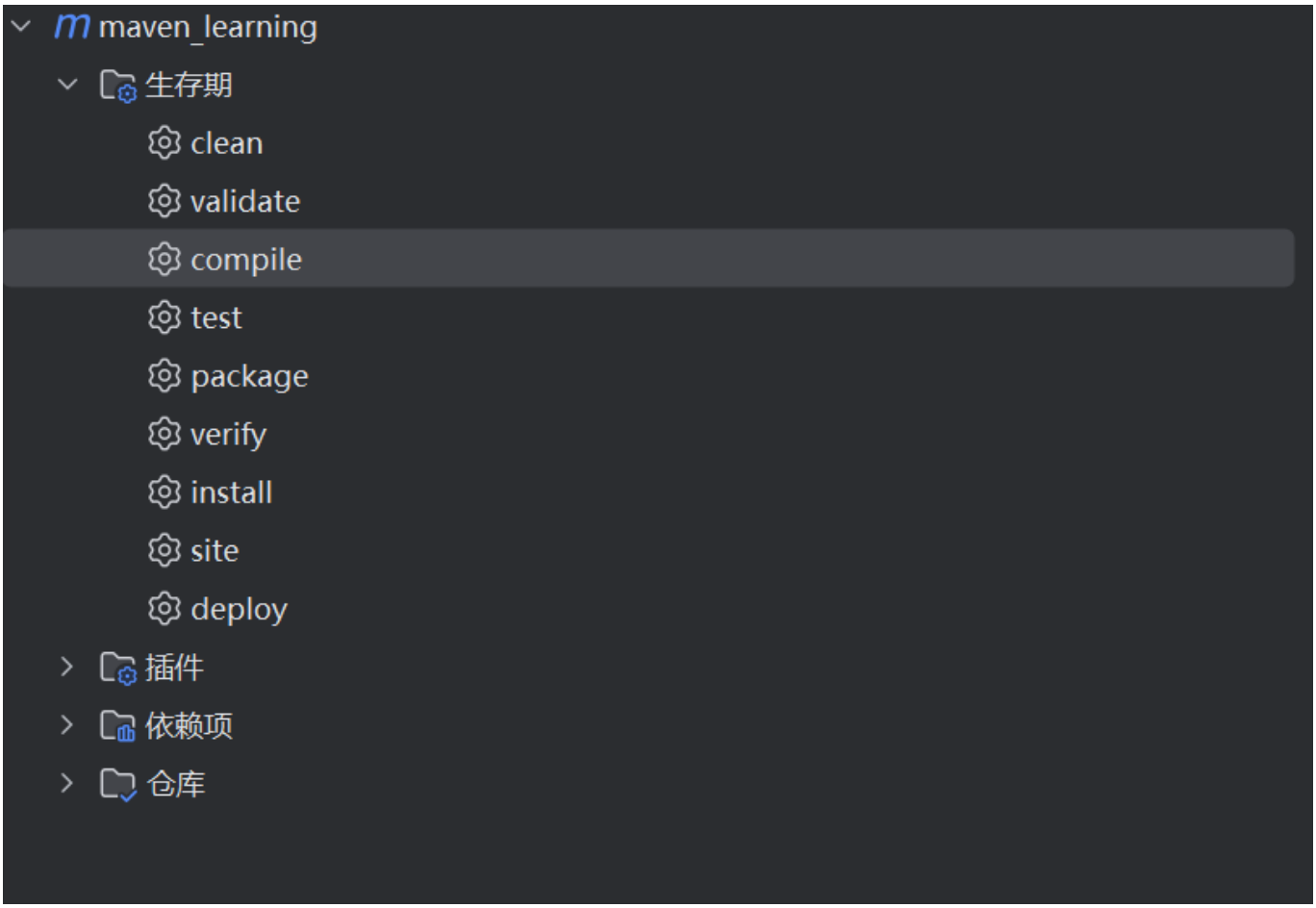
常用参数

- `-Dproperty=value`：定义系统属性，如上文提到的 `-DskipTests`。
- `-P profile-id`：激活指定的构建剖面（Profile）。例如：`mvn install -Pprod,integration`。

> **Profile** 是一个可选的、条件化的构建配置块。它包含了一组可以在特定情况下被激活的配置，用来覆盖或补充 POM 文件中的默认配置。它能让项目适应不同的运行环境。

- `-X` 或 `--debug`：以调试模式运行，输出详细的执行日志。
- `-e` 或 `--errors`：显示完整的错误堆栈信息。
- `-q` 或 `--quiet`：静默模式，只输出错误信息。
- `-f` 或 `--file`：指定另一个 POM 文件，例如：`mvn -f sub-module/pom.xml clean install`。
- `-pl`：指定要构建的模块，例如：`mvn install -pl module-a`。
- `-am`：同时构建 `-pl` 所列模块依赖的模块。

5.2 IDEA中的maven面板可以快速执行哪些maven指令



> mvn validate的作用是验证项目描述（POM文件）的正确性和完整性，这个阶段发生在任何编译、测试或打包工作之前。

> mvn verify的作用是运行集成测试、执行静态代码分析、检查测试覆盖率，这个阶段发生在代码已经编译完成、测试通过、并且打包成功之后。

Task2.快递取件码查询系统

HTTP常见请求方式及常见响应状态码

> 参考菜鸟教程（HTTP）

请求方式

- `GET`：从服务器获取资源。用于请求数据而不对数据进行更改。例如，从服务器获取网页、图片等。
- `POST`：向服务器发送数据以创建新资源。常用于提交表单数据或上传文件。发送的数据包含在请求体中。

- `PUT`：向服务器发送数据以更新现有资源。如果资源不存在，则创建新的资源。与 POST 不同的是，多次执行相同的 PUT 请求不会产生不同的结果。
- `DELETE`：从服务器删除指定的资源。请求中包含要删除的资源标识符。
- `PATCH`：对资源进行部分修改。与 PUT 类似，但 PATCH 只更改部分数据而不是替换整个资源。
- `HEAD`：类似于 GET，但服务器只返回响应的头部，不返回实际数据。用于检查资源的元数据（例如，检查资源是否存在，查看响应的头部信息）。
- `OPTIONS`：返回服务器支持的 HTTP 方法。用于检查服务器支持哪些请求方法，通常用于跨域资源共享（CORS）的预检请求。

> **跨域资源共享（CORS）**这个机制的作用是：允许一个网页从不同的源（域名、协议、端口）请求资源，从而绕过了浏览器的同源策略限制。

> **同源策略**是浏览器的一个核心安全功能。它规定：一个源的文档或脚本，在没有明确授权的情况下，不能与另一个源的资源进行交互。

> **同源** 指的是：协议、域名、端口 三者必须完全相同。

- `TRACE`：回显服务器收到的请求，主要用于诊断。客户端可以查看请求在服务器中的处理路径。
- `CONNECT`：建立一个到服务器的隧道，通常用于 HTTPS 连接。客户端可以通过该隧道发送加密的数据。

状态码

HTTP 状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型。响应分为五类：信息响应(100–199)，成功响应(200–299)，重定向(300–399)，客户端错误(400–499)和服务器错误 (500–599)：

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

HTTP状态码列表:

状态码	状态码英文名称	中文描述
100	Continue	继续。客户端应继续其请求
101	Switching Protocols	切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议
200	OK	请求成功。一般用于GET与POST请求
201	Created	已创建。成功请求并创建了新的资源
202	Accepted	已接受。已经接受请求，但未处理完成

状态码	状态码英文名称	中文描述
203	Non-Authoritative Information	非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本
204	No Content	无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档
205	Reset Content	重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域
206	Partial Content	部分内容。服务器成功处理了部分GET请求
300	Multiple Choices	多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择
301	Moved Permanently	永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
302	Found	临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI
303	See Other	查看其它地址。与301类似。使用GET和POST请求查看
304	Not Modified	未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源
305	Use Proxy	使用代理。所请求的资源必须通过代理访问
306	Unused	已经被废弃的HTTP状态码
307	Temporary Redirect	临时重定向。与302类似。使用GET请求重定向
400	Bad Request	客户端请求的语法错误，服务器无法理解
401	Unauthorized	请求要求用户的身份认证
402	Payment Required	保留，将来使用
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求
404	Not Found	服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面
405	Method Not Allowed	客户端请求中的方法被禁止
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求

状态码	状态码英文名称	中文描述
407	Proxy Authentication Required	请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权
408	Request Time-out	服务器等待客户端发送的请求时间过长，超时
409	Conflict	服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突
410	Gone	客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置
411	Length Required	服务器无法处理客户端发送的不带Content-Length的请求信息
412	Precondition Failed	客户端请求信息的先决条件错误
413	Request Entity Too Large	由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息
414	Request-URI Too Large	请求的URI过长（URI通常为网址），服务器无法处理
415	Unsupported Media Type	服务器无法处理请求附带的媒体格式
416	Requested range not satisfiable	客户端请求的范围无效
417	Expectation Failed（预期失败）	服务器无法满足请求头中 Expect 字段指定的预期行为。
418	I'm a teapot	状态码 418 实际上是一个愚人节玩笑。它在 RFC 2324 中定义，该 RFC 是一个关于超文本咖啡壶控制协议（HTCPCP）的笑话文件。在这个笑话中，418 状态码是作为一个玩笑加入到 HTTP 协议中的。
500	Internal Server Error	服务器内部错误，无法完成请求
501	Not Implemented	服务器不支持请求的功能，无法完成请求
502	Bad Gateway	作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应

状态码	状态码英文名称	中文描述
503	Service Unavailable	由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的
504	Gateway Time-out	充当网关或代理的服务器，未及时从远端服务器获取请求
505	HTTP Version not supported	服务器不支持请求的HTTP协议的版本，无法完成处理

其中，常见的状态码：

- 200 OK
- 301 Moved Permanently
- 302 Found
- 304 Not Modified
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout

JSON数据格式和TCP/IP 通信

主要通过以下网站学习了解：

菜鸟教程（JSON）

JSON: JavaScript Object Notation(JavaScript 对象表示法)

菜鸟教程（TCP/IP）

TCP/IP 是基于 TCP 和 IP 这两个最初的协议之上的不同的通信协议的大集合。

项目实现

做题之前还恶补了一下java网络编程：

Java 网络编程

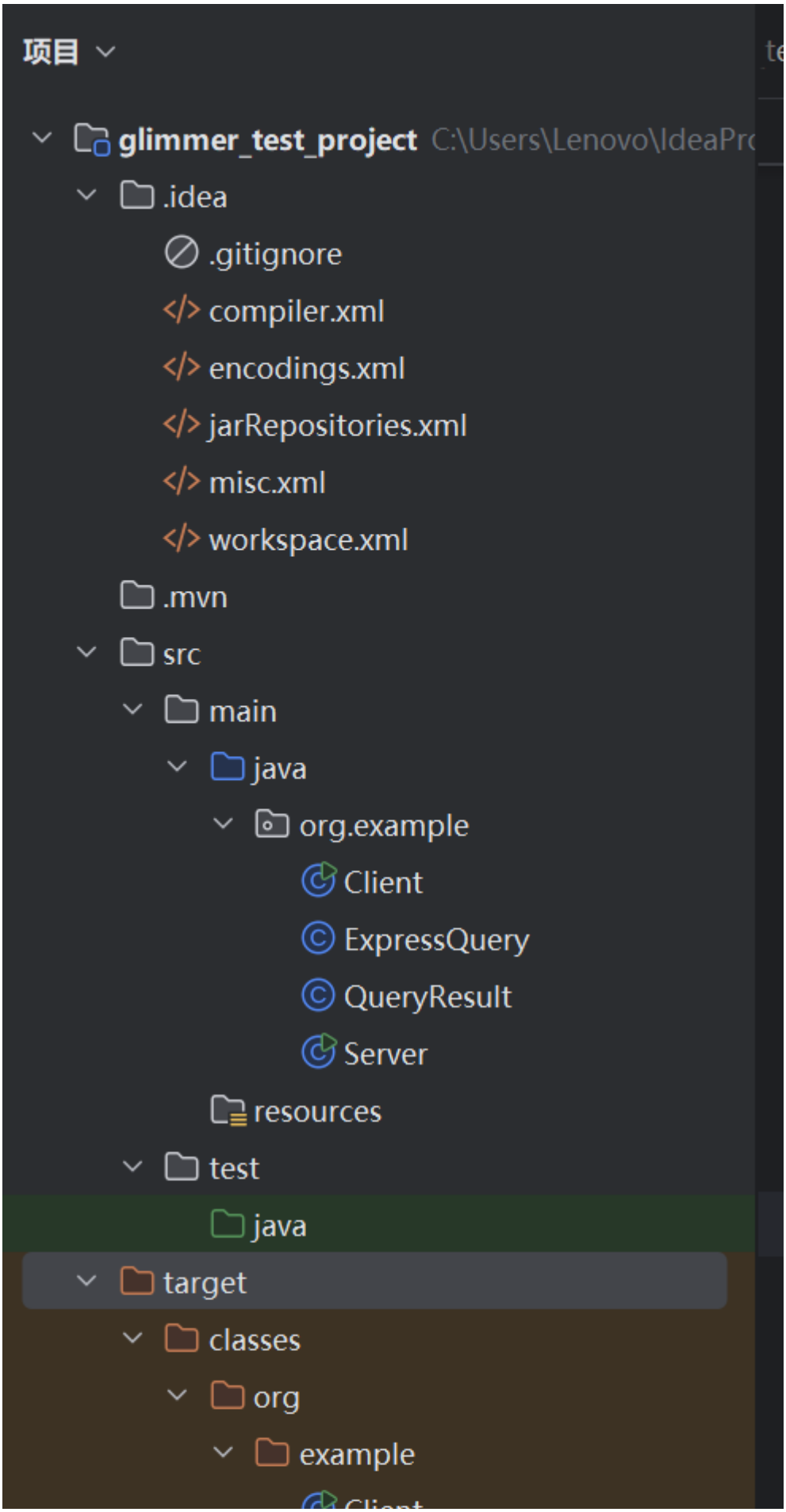
【Java基础】- HttpURLConnection详解

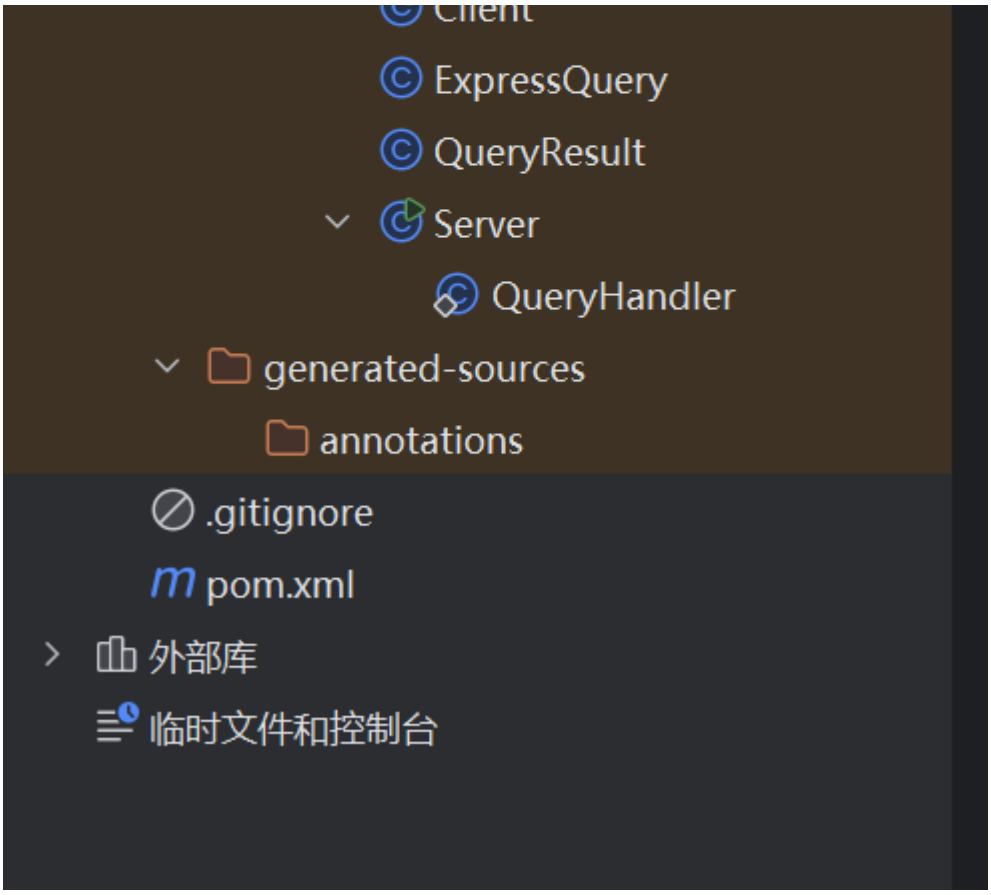
JDK 内置的轻量级 HTTP 服务器 --- HttpServer

FastJson使用详解这一篇就够了

b站教程

maven项目结构





pom.xml文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>

    <groupId>org.example</groupId>
    <artifactId>glimmer_test_project</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>24</maven.compiler.source>
        <maven.compiler.target>24</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>2.0.51</version>
        </dependency>
    </dependencies>
</project>
```

fastjson映射类

```
package org.example;

public class ExpressQuery {
    private String trackingNumber;
    private String phone;

    public ExpressQuery() {}

    public String getTrackingNumber() {
        return trackingNumber;
    }

    public void setTrackingNumber(String trackingNumber) {
        this.trackingNumber = trackingNumber;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

```
package org.example;

public class QueryResult {
    private String pick_code;
    private String msg;

    public QueryResult() {}

    public String getPick_code() {
        return pick_code;
    }

    public void setPick_code(String pick_code) {
        this.pick_code = pick_code;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
```

```
    }
}
```

客户端

```
package org.example;

import com.alibaba.fastjson.JSON;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

public class Client {
    private static final String SERVER_URL = "http://localhost:8000/query";

    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        ExpressQuery query=new ExpressQuery();
        while (true) {
            //读取用户输入的快递单号和手机号
            System.out.print("请输入快递单号: ");
            String trackingNumber=scanner.nextLine();
            System.out.print("请输入手机号: ");
            String phone=scanner.nextLine();

            try {
                /*
                用fastjson这个依赖构造json格式，构造的样例为
                {"trackingNumber":"SF123456789","phone":"19867653558"}
                */

                query.setTrackingNumber(trackingNumber);
                query.setPhone(phone);

                String json= JSON.toJSONString(query);

                // 将这个json格式的数据写入请求体并发送HTTP POST请求

                URL url=new URL(SERVER_URL);
                HttpURLConnection connection=(HttpURLConnection)
url.openConnection();
                connection.setConnectTimeout(3000);// 限制socket等待建立连接的时间，
超时将会抛出java.net.SocketTimeoutException
                connection.setReadTimeout(3000);// 限制输入流等待数据到达的时间，超时
将会抛出java.net.SocketTimeoutException
                connection.setRequestMethod("POST");// 设定请求的方法为"POST"，默认
是GET
                connection.setRequestProperty("Content-
```

```
Type","application/json;charset=utf-8");// 设置传送的内容类型是json格式

connection.setRequestProperty("Accept","application/json;charset=utf-8");// 接收的内容类型也是json格式
                connection.setDoInput(true);// 设置是否从httpUrlConnection读入，默认情况下是true
                connection.setDoOutput(true);// 由于URLConnection在默认的情况下不允许输出，所以在请求输出流之前必须调用setDoOutput(true)。为一个HTTP URL将doOutput设置为true时，请求方法将由GET变为POST
                connection.setUseCaches(false);// 是否使用缓存，Post方式不能使用缓存

                OutputStream os=connection.getOutputStream();
                os.write(json.getBytes(StandardCharsets.UTF_8));
                os.flush();

                connection.connect();

                // 读取响应，状态码是200才是响应成功
                if (connection.getResponseCode() == 200) {
                   /*
                   解析服务端响应的json格式，拿到取件码，拿不到就显示msg的内容
                   样例:
                   例如: {"pick_code":4096,"msg":"success"}
                        {"pick_code":null,"msg":"手机号不正确"}
                   */
                    InputStream is=connection.getInputStream();
                    StringWriter sw=new StringWriter();
                    InputStreamReader isr=new
InputStreamReader(is,StandardCharsets.UTF_8);
                    char[] buffer=new char[1024];
                    int len;
                    while((len=isr.read(buffer))!=-1){
                        sw.write(buffer,0,len);
                    }
                    String res=sw.toString();
                    QueryResult result=JSON.parseObject(res,QueryResult.class);

                    if(result.getPick_code()==null){
                        System.out.println("查询失败，信息:  "+result.getMsg());
                    } else {
                        System.out.println("查询成功，取件码:
"+result.getPick_code());
                    }

                    isr.close();
                    sw.close();
                    is.close();

                } else {
                    System.out.println("查询失败，状态码: " +
connection.getResponseCode());
                }

                connection.disconnect();
```

```
                os.close();
            } catch (IOException e) {
                System.out.println("请求发生错误: " + e.getMessage());
            }
        }
//          下面这两行不注释掉的话，IDEA无法运行该程序
//          会显示无法到达该语句
//          scanner.close();
//          System.out.println("客户端已退出");
    }


}
```

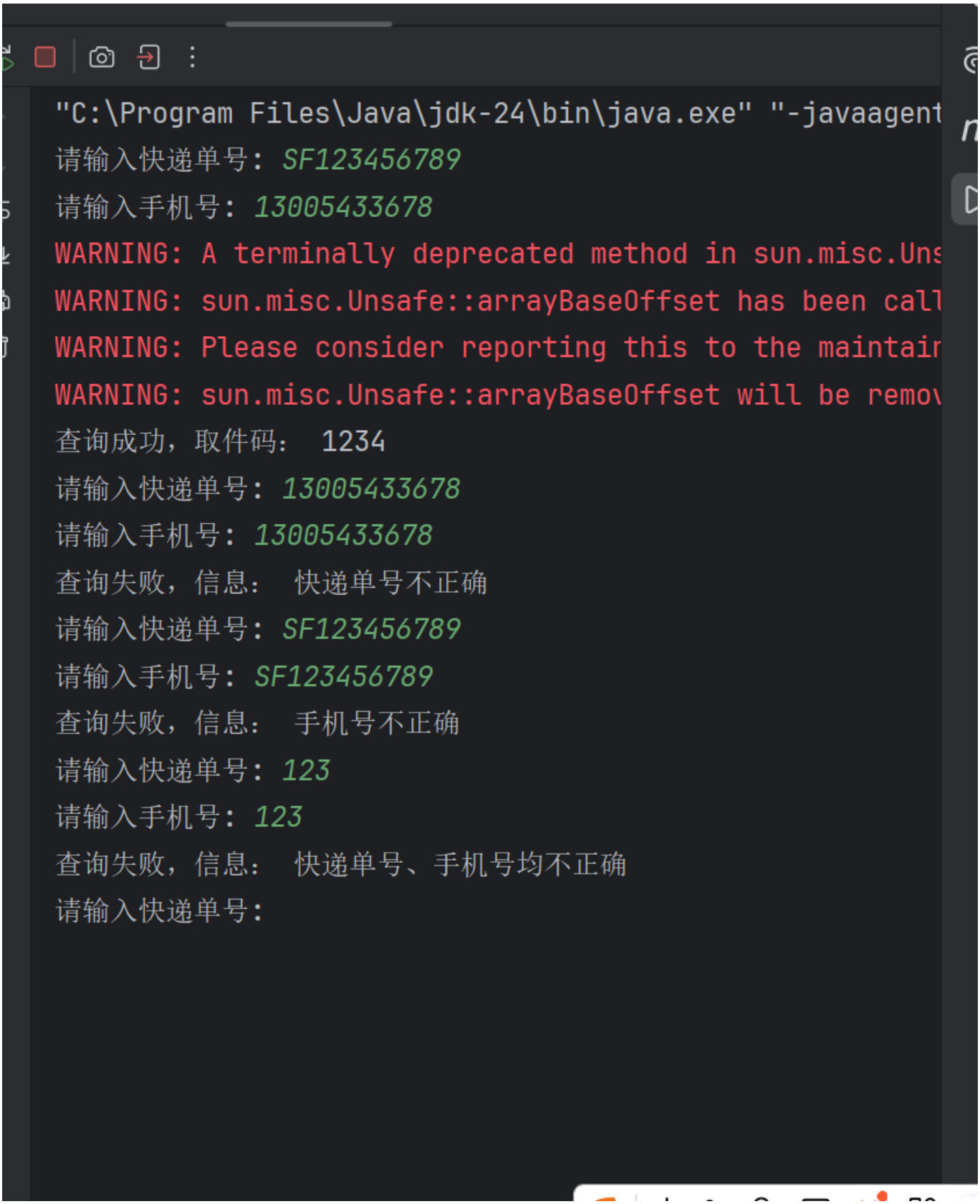
服务端

```
package org.example;

import com.alibaba.fastjson.JSON;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

import java.io.*;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;

public class Server {
    private static final int PORT = 8000;
    private static final Map<String, String> expressMap = new HashMap<>();

    public static void main(String[] args) throws IOException {
        // 初始化一些测试数据
        initializeExpressData();
        // 创建HTTP服务器，监听指定端口
        HttpServer server = HttpServer.create(new InetSocketAddress(PORT), 0);
        // 设置路由和处理程序
        server.createContext("/query", new QueryHandler());
        // 启动服务器
        server.start();
        System.out.println("Server started on port " + PORT);
    }

    private static void initializeExpressData() {
        // 添加一些测试数据
        //键的构成是  快递单号_手机号
        expressMap.put("SF123456789_13005433678", "1234");
        expressMap.put("JD987654321_19805433168", "5678");
        expressMap.put("YT456789123_13905479698", "9012");
```

```
        expressMap.put("ZT789123456_18505433664", "3456");
    }

    static class QueryHandler implements HttpHandler {
        @Override
        public void handle(HttpExchange exchange) throws IOException {
            if ("POST".equals(exchange.getRequestMethod())) {
                // 读取请求体
                InputStream is=exchange.getRequestBody();
                StringWriter sw=new StringWriter();
                InputStreamReader isr=new
InputStreamReader(is,StandardCharsets.UTF_8);
                char[] buffer=new char[1024];
                int len;
                while((len=isr.read(buffer))!=-1){
                    sw.write(buffer,0,len);
                }
                String json=sw.toString();

//                调试用
//                System.out.println("读取请求体: "+json);

                try {
                    // 解析JSON请求（例如：{"trackingNumber":"SF123456789"}）获得单
号
                    ExpressQuery query= JSON.parseObject(json,ExpressQuery.class);
                    String trackingNumber=query.getTrackingNumber();
                    String phone= query.getPhone();
                    String key=trackingNumber+"_"+phone;


                    // 在expressMap中查询取件码,
                    QueryResult result=new QueryResult();
                    if(expressMap.containsKey(key)){
                        result.setMsg("success");
                        result.setPick_code(expressMap.get(key));
                    }else{
                        result.setPick_code(null);
                        result.setMsg("快递单号、手机号均不正确");
                        for(String aKey : expressMap.keySet()){
                            String[] keys=aKey.split("_");
                            if(keys[0].equals(trackingNumber)){
                                result.setMsg("手机号不正确");
                                break;
                            }
                            if(keys[1].equals(phone)){
                                result.setMsg("快递单号不正确");
                                break;
                            }
                        }
                    }

                    /*
                    构建响应的json
```

```
                    例如：{"pick_code":4096,"msg":"success"}
                    如果找不到快递则是{"pick_code":null,"msg":"根据各种情况写提示信
息"}
                    */
                    String resJson=JSON.toJSONString(result);

//                    调试用
//                    System.out.println("构建响应：  "+resJson);

                    // 发送响应
                    exchange.getResponseHeaders().add("Content-
Type","application/json;charset=utf-8");
                    exchange.sendResponseHeaders(200,
(resJson.getBytes(StandardCharsets.UTF_8)).length);

//                    注意！！！！！！
//                    是  Response  ，不是  Request  ！
//                    自动补全害人不浅！

                    OutputStream os=exchange.getResponseBody();
                    os.write(resJson.getBytes(StandardCharsets.UTF_8));

                    os.close();
                    exchange.close();

                    System.out.println("发送响应");

                } catch (Exception e) {

//                    调试用
//                    System.out.println("处理请求时发生错误：  " + e.getMessage());
//                    e.printStackTrace();

                    // 处理异常，返回400状态码(Bad Request)
                    exchange.sendResponseHeaders(400,-1);

                }
            } else {
                // 非POST请求返回405 Method Not Allowed
                exchange.sendResponseHeaders(405, -1);
            }
        }
    }
}
```

运行结果截图


"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent
请输入快递单号：SF123456789
请输入手机号：13005433678
WARNING: A terminally deprecated method in sun.misc.Uns
WARNING: sun.misc.Unsafe::arrayBaseOffset has been call
WARNING: Please consider reporting this to the maintair
WARNING: sun.misc.Unsafe::arrayBaseOffset will be remov
查询成功，取件码： 1234
请输入快递单号：13005433678
请输入手机号：13005433678
查询失败，信息： 快递单号不正确
请输入快递单号：SF123456789
请输入手机号：SF123456789
查询失败，信息： 手机号不正确
请输入快递单号：123
请输入手机号：123
查询失败，信息： 快递单号、手机号均不正确
请输入快递单号：