Java-10

一、Task1.Maven

1.什么是maven?

1. **Maven** 是一个用于项目管理和构建的自动化工具，主要用于 Java 项目
2. **jar包(Java Archive):** 是一个压缩包，包含了运行java程序的文件和数据
    - 库 **jar包:** 提供类和方法供其他 Java 项目使用
    - 可执行 **jar包:** 可以直接运行的程序
3. **两者关系: Maven** 管理并生成 **jar包**
    - 在项目构建过程中，**Maven** 可以从远程仓库下载管理需要的 **jar包**
    - 项目开发完成之后 **Maven** 可以将其打包成 **jar包**
4. 为什么使用 **Maven:**
    - 不同的编译工具可能导致 **Java项目** 的结构不同，**Maven** 统一结构
    - 防止项目需要的 **jar包** 被包含在程序中导致每个项目占据更大的存储空间
    - 防止 **jar包** 之间产生版本冲突需要手动升降版本问题

2. 试着下载maven吧!

- 配置maven类似配置Java环境
-
```
C:\Users\AyaU>mvn -v
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
Maven home: F:\Maven\apache-maven-3.9.11
Java version: 24.0.2, vendor: Oracle Corporation, runtime: C:\Program Files
Default locale: zh_CN, platform encoding: UTF-8
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
```

3.什么是maven仓库?

1. 本地仓库位置:
    - 按照路径**apache-maven-3.9.11\conf\settings.xml**

      打开文件 **settings.xml**
    -
```
<!--  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>F:\repository</localRepository>
```

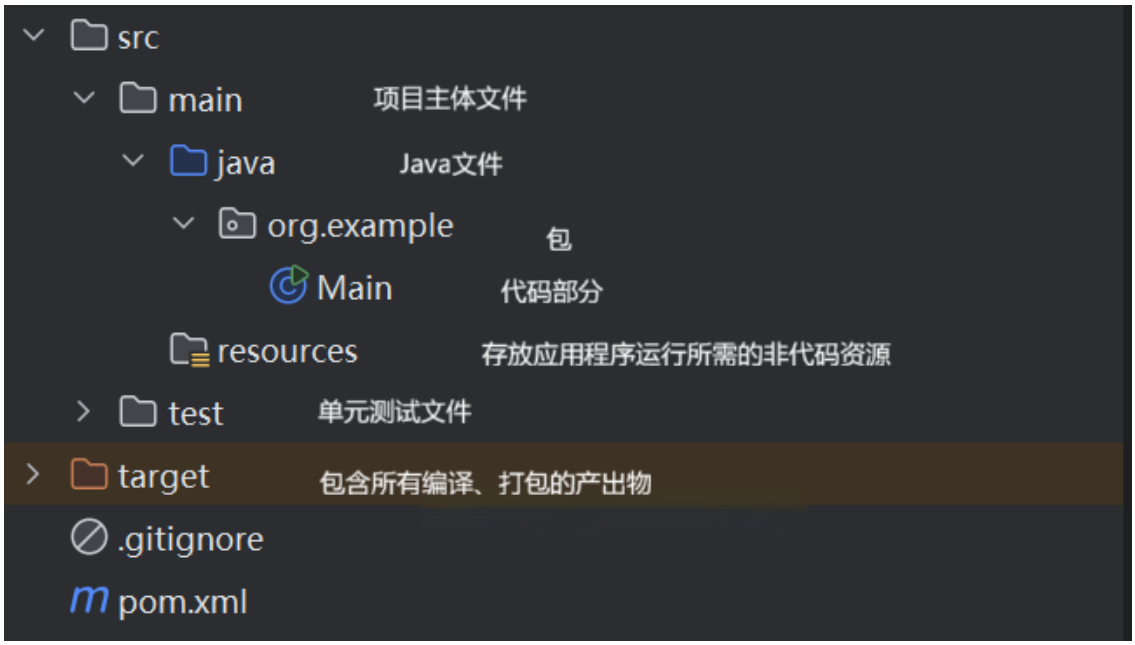
      复制/path/to/local/repo到注释块外，并修改成本地仓库地址
2. 远程仓库:
    - **中央仓库** 如果不在 **pom.xml** 中配置其他仓库，就只从这里下载
    - **其他公共仓库**
    - **私服(私有仓库)**
3. 私服:即公司内部仓库

   实际开发中，通常不会直接连接**中央仓库**，而是先访问**私服**，再考虑**中央仓库**

   私服是离线可用的，下载速度也快得多

4.创建你自己的maven项目!

1. 项目需要配置的参数:
    - **groupId** 组织/公司标识
    - **artifactId** 项目名称
    - **version** 项目版本
2. 结构:



3. pom.xml文件(Project Object Model):
    - 是项目的配置文件，包含文件的一切基本信息
    - 作用:
        - 项目名称、版本、描述
        - 声明项目依赖 如: **jar包**
        - 构建配置
        - 环境配置 如:JDK版本 仓库位置 环境变量配置
    - 管理jar包版本和项目类型:
        - jar包版本: **pom.xml** 的 后修改依赖
        - 项目类型(打包方式): 在文件中找到

          jar

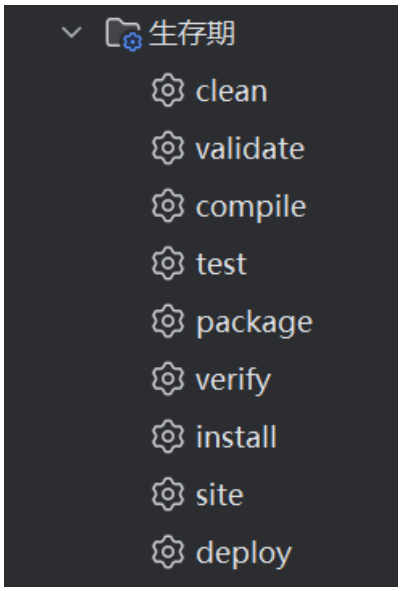
          更改类型 **(如jar)** 即可

          支持的主要类型:

          - **jar(默认类型)**
          - **war(web应用程序)** 需要部署到外部容器的应用
          - **pom(父项目)** 不生成实际jar/war，用于管理子模块
4. 导入 **jar包**:
    - 查找 **jar包 依赖坐标 (官方文档/MVNRepository/Idea集成)**
    - 将坐标添加到 **pom.xml** 的 后

5.启动你的maven！

1. 常用命令:
    - **clean** 删除 **target** 目录
    - **compile** 编译源代码到 **target/classes** 目录
    - **test** 运行所有测试样例
    - **package** 编译、测试，并将代码打包成 jar/war 文件到 **target** 目录
    - **install** 将项目构建并安装到本地 Maven 仓库
    - **deploy** 将最终的项目包复制到远程仓库
2. Idea执行:



二、Task2.快递取件码查询系统

1. **HTTP:**
    - 请求:
        1. 请求行:方法/地址**(结合下面的HOST)**/版本号，如:

```
GET /hello.txt HTTP/1.1
```

        2. 请求方法:
            - **GET:**请求指定页面信息，并返回实体主体
            - **POST**:向指定资源提交数据，可能导致新资源建立/已有资源修改
            - **PUT:**传送数据替换目标资源**(提供完整的资源)**
            - **DELETE:** 删除指定的资源
            - **PATCH:**对资源应用部分修改**(只发送变化部分)**
            - **HEAD:**与**GET**相同，但只返回报头，不返回响应体(检查存在)
            - **OPTIONS:**获取服务器支持的请求方法
            - **TRACE:**服务器返回收到的请求消息(测试诊断)
            - **CONNECT:**建立一个隧道用于代理服务器的通信，通常用于 HTTPS
        3. 请求头:
            - **Host:**指定请求的目标服务器和端口号
            - **Connection:**客户端与服务连接类型**(keep-alive/close)**
            - **Upgrade-Insecure-Requests:**升级为HTTPS请求
            - **User-Agent:**浏览器的详细信息标识
            - **Accept:**传输文件类型

- **Accept-Language:**指出浏览器接受的语言种类
- **Accept-Encoding:**指出浏览器的文件编解码格式
- **Accept-Charset:**声明客户端支持的字符集
- **Cookie:**发送服务器之前设置的Cookie
- **Referer:**表明产生请求的网页来自于哪个 URL(判断是否是本网站的地址)
- **Content-Type:**声明请求体的内容类型

4. 空行:分隔头部和消息体
5. 消息体:JSON格式

2. **项目实现:**
    1. 导入解析json格式的依赖(pom文件中)

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

    2. 构造json格式

       必须有无参构造方法

    3. 客户端请求的代码实现:
        - 使用 HttpURLConnection 类

          由connection.getResponseCode()可知也是题目中结构所需

            1. 先建立连接

```
URL url = new URL(SERVER_URL);
HttpURLConnection connection
    = (HttpURLConnection) url.openConnection();
```

            2. 设置方法和参数 **(设置DoOutput)**

```
connection.setRequestMethod("POST");
//Content-Type有请求体时必须设置
connection.setRequestProperty
    ("Content-Type", "application/json");
//Accept不是必要设置的，但最好设置，它告诉服务器希望接收什么格式的响应
connection.setRequestProperty
    ("Accept", "application/json");
```

            3. 输出

```

```

        - 使用 HttpClient 库
        - 使用 Okhttp 库
        - 使用 Spring 库的 RestTemplate

4. 服务端接收实现:
    1. 整体思路:读取请求体-设置响应头-设置响应体
    2. 读取请求体的过程中，

       我错误使用`String request = exchange.getRequestURI().getQuery();`

       事实上这是读取了**URL中问号(?)后面的整个查询字符串**，而非**请求体**
5. 报错:

   输入不存在的数据时出现 **Unexpected end of file from server** 报错

   我锁定错误的位置在于服务端的代码:

   ```
   try (OutputStream os = exchange.getResponseBody())
   {os.write(responseBytes); }
   ```

   这里抛出了错误导致了报错

   但是，为什么会无法构建这个流了呢?

   不断试错和查阅资料之后我找到了问题所在

   首先 我使用:

   ```
   exchange.sendResponseHeaders(200, response.length());
   ```

   去构建 **header**，其中的 **response.length()** 就是关键

   它返回的是字符数，不是字节数，对于包含**中文字符(来源于错误提示，正确提示没有中文，因此没有报错)**的响应，UTF-8 编码下字符数和字节数不一致，服务器发送的字节数少于声明的长度，就关闭了连接

   因此，解决这个问题就只需要转成**使用字节数组计算长度**的方式即可

   ```
   byte[] responseBytes = response.getBytes(StandardCharsets.UTF_8);
   Headers headers = exchange.getResponseHeaders();
   headers.set("Content-Type", "application/json;charset=utf-8");
   exchange.sendResponseHeaders(200, responseBytes.length);
   ```

   像这样构建响应头即可
6. 代码实现:
    - **class**

      ```
      public class User {
          public String trackingNumber;
          public String phone;

          public User() {
          }

          public User(String trackingNumber, String phone) {
              this.trackingNumber = trackingNumber;
              this.phone = phone;
          }

          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;
    }
}
```

- **客户端**

```
import com.alibaba.fastjson.JSON;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URI;
import java.net.URL;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
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);
    boolean flag = true;

    while (true) {
    if (flag == false) {
    //询问是否退出,且在至少循环一次之后
    System.out.println("输入exit退出，否则继续取件");
    String request = scanner.nextLine();
    if (request.equals("exit")) {
        break;
                }
            }
    flag = false;

    //读取用户输入的快递单号和手机号
    System.out.println("请输入快递单号:");
    String trackingNumber = scanner.nextLine();
    System.out.println("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"}*/
        User user =  new User(trackingNumber, phone);
        String json = JSON.toJSONString(user);

        // 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.setRequestMethod("POST");
        //Content-Type有请求体时必须设置
        connection.setRequestProperty("Content-Type",
    "application/json");
        //Accept不是必要设置的，但最好设置，它告诉服务器希望接收什么格式的响应
        connection.setRequestProperty("Accept", "application/json");
        connection.setDoOutput(true);

        try (OutputStream os = connection.getOutputStream()) {
                        os.write(json.getBytes(StandardCharsets.UTF_8));
                    }

         // 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"} */
         try (BufferedReader br =
              new BufferedReader(new InputStreamReader
              (connection.getInputStream(),
              StandardCharsets.UTF_8)))
            {
              StringBuilder response = new StringBuilder();
              String line;
            while ((line = br.readLine()) != null) {
               response.append(line);}
               System.out.println(response);
            }
         }
         else
         {
            System.out.println("查询失败，状态码: " +
    connection.getResponseCode());

     System.out.println(connection.getResponseMessage());
         }
    } catch (IOException e) {
                    System.out.println("请求发生错误: " + e.getMessage());
            }
        }
            scanner.close();
            System.out.println("客户端已退出");
    }
    }
```

抱歉格式有点杂乱😣

- **服务端**

```
import com.alibaba.fastjson2.JSON;
import com.sun.net.httpserver.Headers;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.UUID;

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 inputStream = exchange.getRequestBody();
                String request = new
String(inputStream.readAllBytes(), StandardCharsets.UTF_8);

                try {
```

```
                    // 解析JSON请求（例如：
{"trackingNumber":"SF123456789"}）获得单号
                    User user = JSON.parseObject(request,
User.class);
                    String key = user.trackingNumber+"_"+user.phone;

                    // 在expressMap中查询取件码，
                    String code = expressMap.get(key);

                    /*
                    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 response;
                    if (code == null) {
                        response = "
{\"pick_code\":"+null+",\"msg\":\""+"订单号或手机号不正确"+"\"}";
                    }
                    else {
                        response = "
{\"pick_code\":"+code+",\"msg\":\""+"success"+"\"}";
                    }

                    //设置响应头
                    byte[] responseBytes =
response.getBytes(StandardCharsets.UTF_8);
                    Headers headers = exchange.getResponseHeaders();
                    headers.set("Content-Type",
"application/json;charset=utf-8");
                    exchange.sendResponseHeaders(200,
responseBytes.length);

                    //设置响应体
                    try (OutputStream os =
exchange.getResponseBody()) {
                        os.write(responseBytes);
                    }

                } catch (Exception e) {
                    // 处理异常，返回400状态码(Bad Request)
                    String response = "
{\"pick_code\":"+null+",\"msg\":\""+"单号或手机号不存在"+"\"}";
                    byte[] responseBytes =
response.getBytes(StandardCharsets.UTF_8);
                    exchange.sendResponseHeaders(400,
responseBytes.length);
                    try (OutputStream os =
exchange.getResponseBody()) {
                        os.write(responseBytes);
                    }

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

```
        }
    }
}
```

这个不改格式了，服务端改了还更乱了🥹

- **运行效果:**

1. 正常输入

```
请输入快递单号:
SF123456789
请输入手机号:
13005433678
```

结果

```
{"pick_code":1234,"msg":"success"}
输入exit退出，否则继续取件
```

2. 输入错误数据

```
{"pick_code":1234,"msg":"success"}
输入exit退出，否则继续取件

请输入快递单号:
114514
请输入手机号:
123456
```

结果

```
{"pick_code":1234,"msg":"success"}
输入exit退出，否则继续取件

请输入快递单号:
114514
请输入手机号:
123456
{"pick_code":null,"msg":"订单号或手机号不正确"}
输入exit退出，否则继续取件
```

3. 输入exit退出

```
请输入手机号:
123456
{"pick_code":null,"msg":"订单号或手机号不正确"}
输入exit退出，否则继续取件
exit
客户端已退出

进程已结束，退出代码为 0
```