基于基础网络编程的简单服务器管理系统

编写环境： java + springboot + socket

部署环境：CentOS + java8

1.代码编写

服务端：

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServer {
    public static void main(String[] args) throws IOException {
        System.out.println("服务器启动\n");
        // 端口号
        int port = 7000;
        // 在端口上创建一个服务器套接字
        ServerSocket serverSocket = new ServerSocket(port);
        int count = port;
        do{
            // 监听来自客户端的连接
            Socket socket = serverSocket.accept();
            DataInputStream dis = new DataInputStream(
                    new BufferedInputStream(socket.getInputStream()));
            DataOutputStream dos = new DataOutputStream(
                    new BufferedOutputStream(socket.getOutputStream()));

            boolean flag = true;
            while(flag){
                String command;
                try{  //处理客户端断开后报EOFException
                    command = dis.readUTF();
                }catch(EOFException e){
                    System.out.println("连接关闭");
                    break;
                }
                System.out.println("服务器端收到的命令为: " + command);

                //服务器处理逻辑
                //重启/关闭比较特殊
                if(command == "reboot" || command == "poweroff"){
                        dos.writeInt(1);
                        Process ps = Runtime.getRuntime().exec(command);
                }else{
```

```
                    try {
                        Process ps = Runtime.getRuntime().exec(command);
                        dos.writeInt(1);
                    } catch (Exception e) {
                        dos.writeInt(0);
                    }
                }
                //回传客户端
                dos.flush();
            }
            socket.close();

            count--;
        }while(count>0);
        serverSocket.close();
    }
}
```

客户端

```
// controller层
import com.LBModelProject.service.CommandService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

import java.io.IOException;

@Controller
@RequestMapping("/command")
public class CommandController {
    //端口默认7000
    final static int port = 7000;

    @Autowired
    CommandService commandService;

    //简单登录
    @RequestMapping("/enter")
    public String enter(){
        return "enter";
    }

    //处理跳转--有待改善
    @RequestMapping("/control_model")
    public String controlModel(){
        return "control_model";
    }

```

```
    //简单通信--网络编程
    @RequestMapping("/do_command")
    @ResponseBody
    public int doCommand(@RequestParam("host")String host,
                         @RequestParam("command")String command) throws IOException {
        if (host == null || command == null) {
            return 0;
        }
        System.out.println("操作服务器ip : " + host);
        System.out.println("操作指令 : " + command);
        //交由服务端执行
        int res = commandService.doSocket(command, host, port);
        System.out.println("返回结果 : " + res);

        return res;
    }
}
```

```
// service层
import org.springframework.stereotype.Service;

import java.io.*;
import java.net.ConnectException;
import java.net.Socket;

@Service
public class CommandService {

    //给服务器发送指令
    public int doSocket(String command, String host, int port) throws IOException {
        try {
            // 创建一个套接字并将其连接到指定端口号
            Socket socket = new Socket(host, port);

            DataInputStream dis = new DataInputStream(new
BufferedInputStream(socket.getInputStream()));

            DataOutputStream dos = new DataOutputStream(new
BufferedOutputStream(socket.getOutputStream()));

            System.out.println("连接成功\n");

            //处理逻辑--可再封装
            //压入缓存
            dos.writeUTF(command);
            //发送给服务器
            dos.flush();

            //服务器回传
            int result = dis.readInt();

```

```
            socket.close();

            return result;
        }catch (ConnectException e){
            System.out.println("连接失败");
            return 0;
        }
    }
}
```

2.环境部署

CentOS

```
# 直接在yum中查找
yum search java|grep jdk

# 使用yum直接安装--默认安装在 /usr/lib/jvm 目录下
yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel

# 配置环境变量
vim /etc/profile

# profile文件中加入
#set java environment
JAVA_HOME=/user/lib/jvm/java-1.8.0-openjdk-1.8.0.222.b10-0.el6_10.x86_64
JRE_HOME=$JAVA_HOME/jre
CLASS_PATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib
PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
export JAVA_HOME JRE_HOME CLASS_PATH PATH

# 使环境变量生效
source /etc/profile

# 检查java版本
java -version
```

开放端口

```
# 开放对应端口
# -zone 作用域
# -add-port=7000/tcp 添加端口，格式为：端口/通讯协议
# -permanent #永久生效，没有此参数重启后失效
firewall-cmd --add-port=7000/tcp --permanent

# 重启防火墙
firewall-cmd --reload
```

编写启动脚本 javaServer.sh --脚本放置在 java.class 同一目录下

注：开机启动(待解决)

```
#! /bin/bash
# java -cp /usr/local/javaWorkHouse/ SocketServer 代表运行对应目录的.class文件，-cp后接路径，classpath
# > /usr/local/javaWorkHouse/access.log 代表将输出写入对应目录的 access.log 文件
# 2>&1& 代表在后台运行
java -cp /usr/local/javaWorkHouse/ SocketServer > /usr/local/javaWorkHouse/access.log 2>&1&
```

3.springboot程序部署

本地java版本：12

服务器java版本：8

3.1 pom.xml 文件配置

```
<!-- pom.xml文件 -->

<?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>com.LBModelProject</groupId>
  <artifactId>LBModel</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!-- 打jar包前要添加此语句 -->
  <packaging>jar</packaging>

  <name>LBModel</name>
```

```
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.6.RELEASE</version>
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

    <!--springboot-->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

    <!-- 添加jdk版本适配依赖 12->8 -->
    <dependency>
      <groupId>javax.xml.bind</groupId>
      <artifactId>jaxb-api</artifactId>
      <version>2.3.0</version>
    </dependency>
    <dependency>
      <groupId>com.sun.xml.bind</groupId>
      <artifactId>jaxb-impl</artifactId>
      <version>2.3.0</version>
    </dependency>
    <dependency>
      <groupId>com.sun.xml.bind</groupId>
      <artifactId>jaxb-core</artifactId>
      <version>2.3.0</version>
    </dependency>
    <dependency>
      <groupId>javax.activation</groupId>
      <artifactId>activation</artifactId>
      <version>1.1.1</version>
```

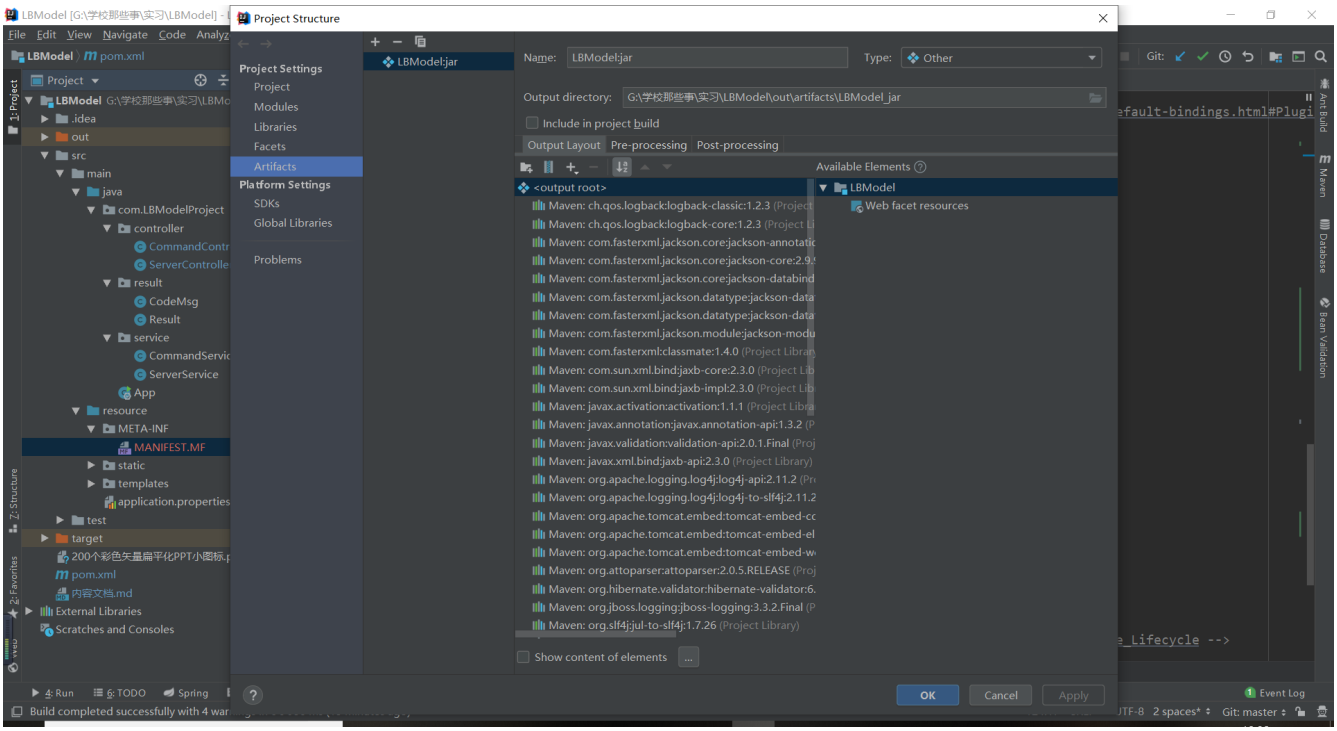
```
    </dependency>
  </dependencies>


  <build>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
      <plugins>

        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>

        <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle -->
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin_bindings_for_jar_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.0.2</version>
          <!-- 添加程序入口路径 -->
          <configuration>
            <archive>
              <manifest>
                <mainClass>com.LBModelProject.App</mainClass>
              </manifest>
            </archive>
          </configuration>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
```

```
          <version>2.8.2</version>
        </plugin>
        <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle -->
        <plugin>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.7.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
          <version>3.0.0</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

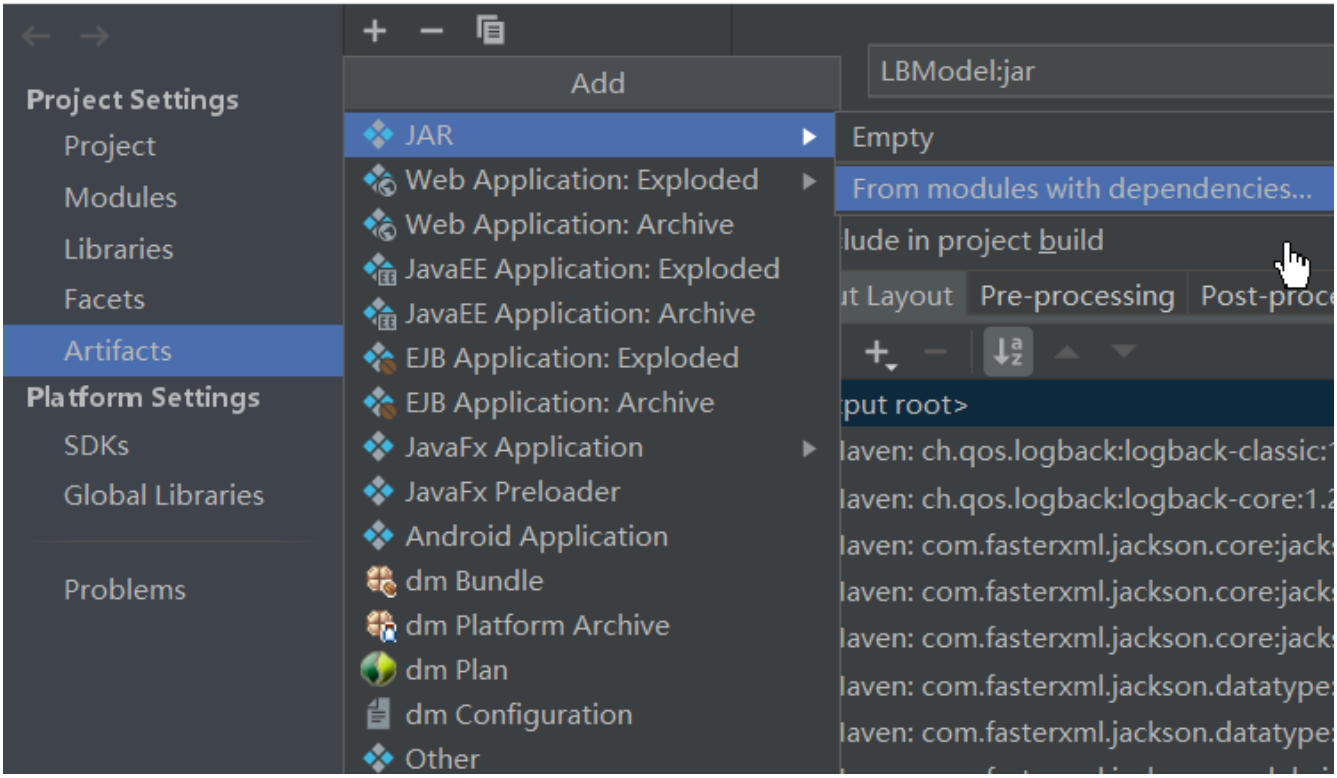
3.2开始打jar包

使用普通方式打jar包（也可直接使用maven的指令打）

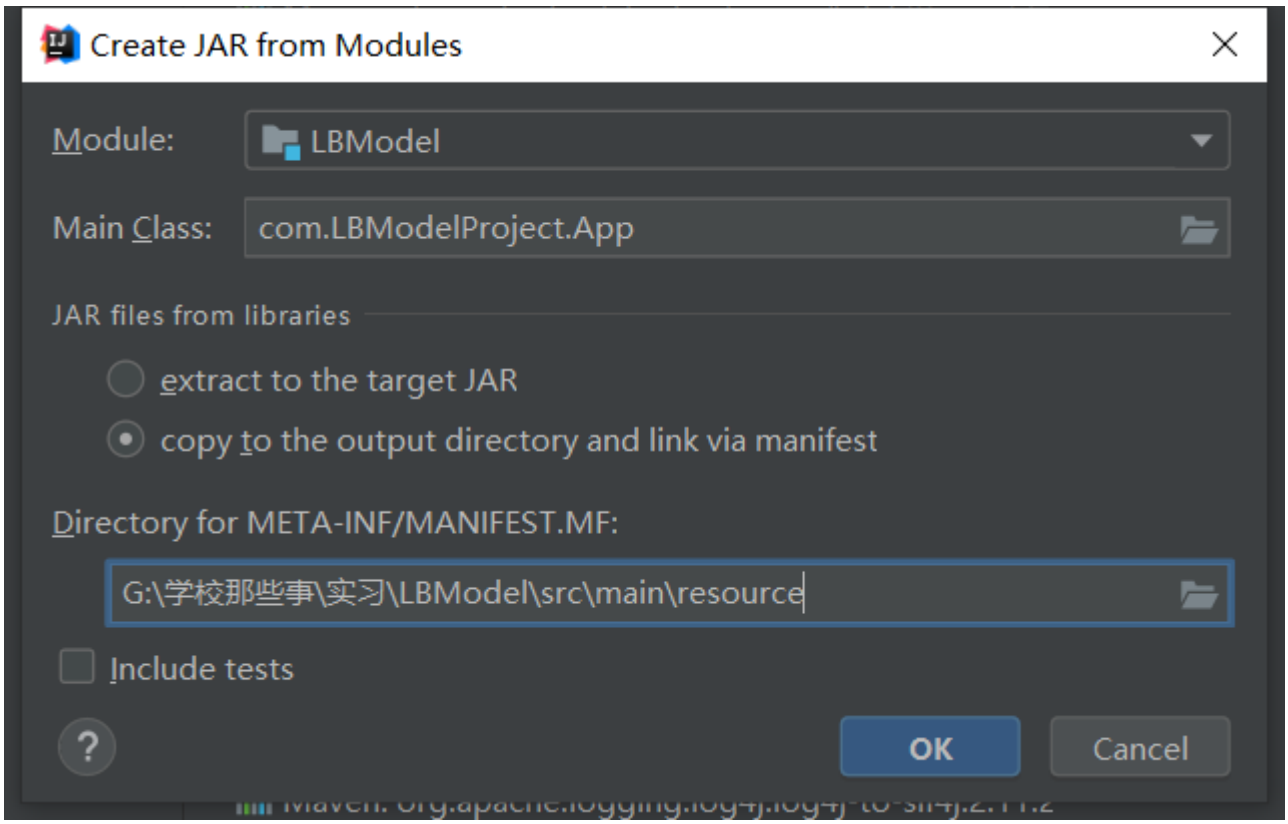
第一步：在file目录下选择Project Structure，选中Artifacts



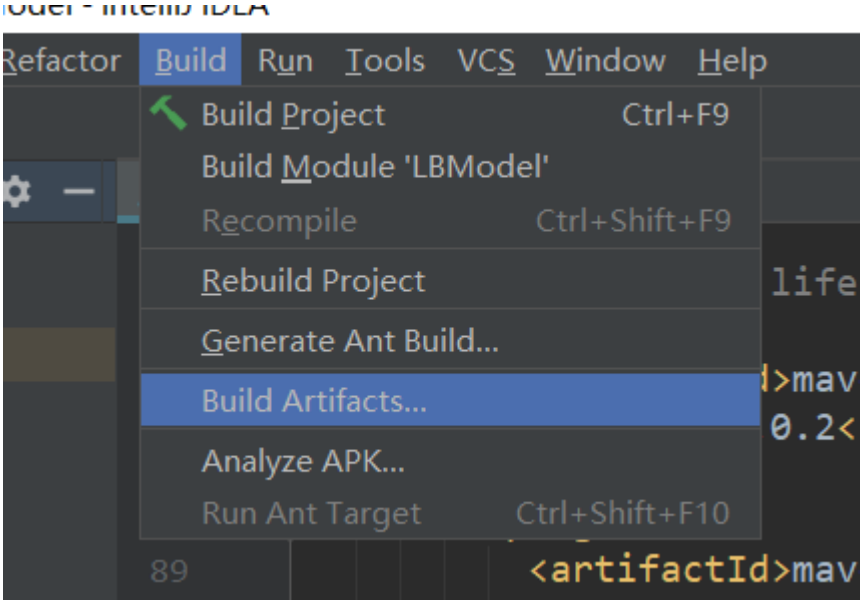
第二步：选中“+”号，指向 JAR，选中From modules with dependencies...

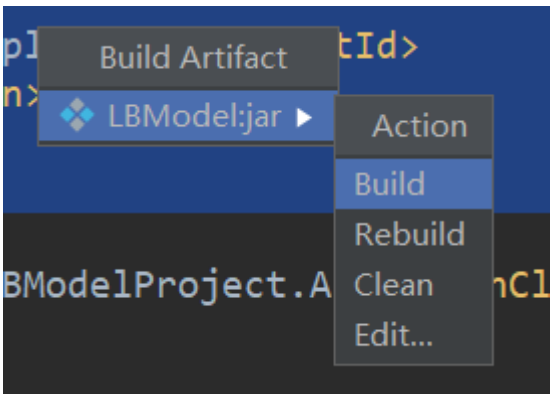


第三步：指定Main Class，即指定主入口（未选后续指向jar包时会报找不到主入口的错误），选copy to the output directory.....选项，接着指定对应的 META-INF/MANIFEST.MF 的存放路径，点击OK后再resource下生产目录和文件

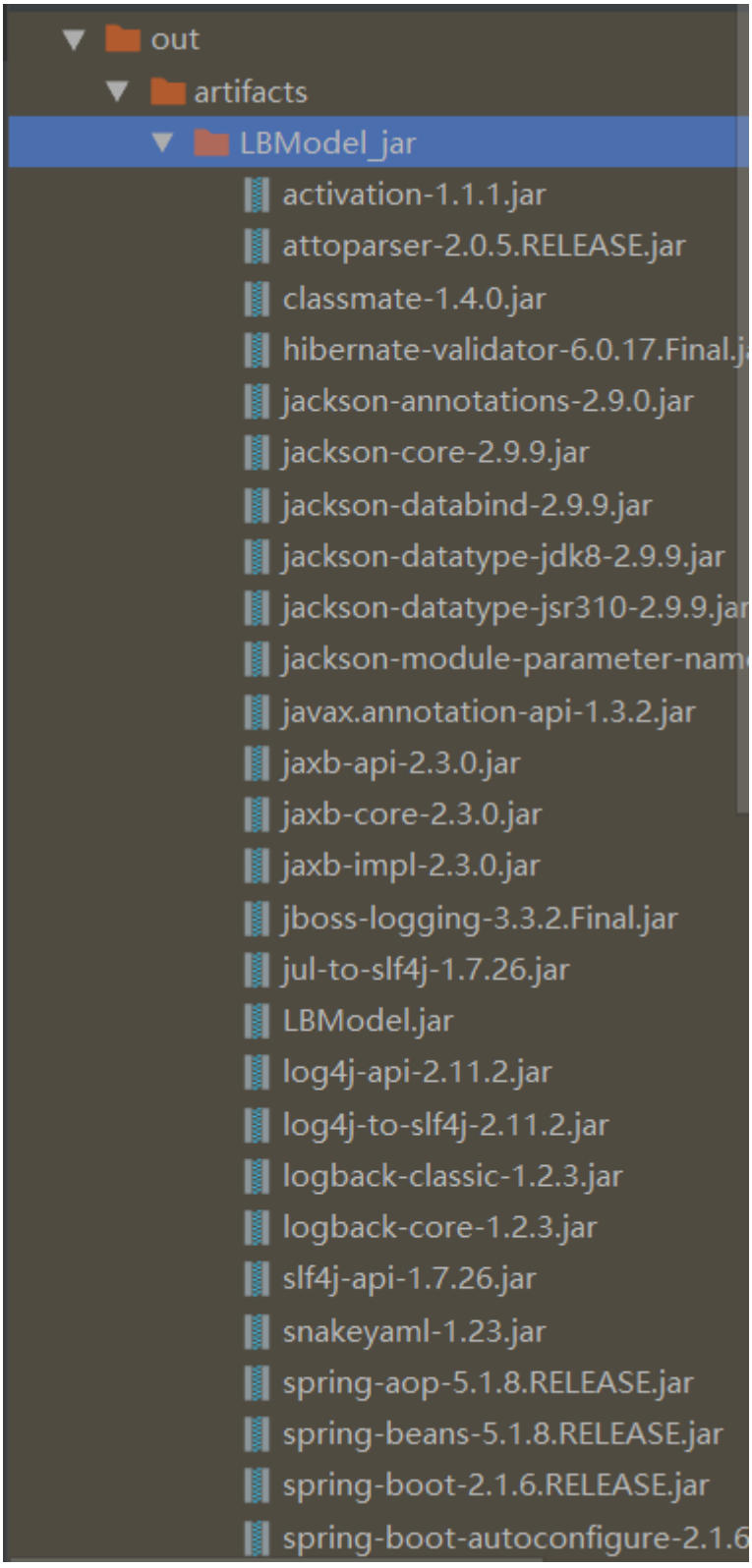


第四步：Apply后继续OK退出Project Structure，回到主界面，选择Build目录，选中Build Artifacts...选项，弹出图二的小框，第一次创建使用Build，后续再次创建可以直接使用Rebuild，不用再次配置Project Structure，除非结构改变。

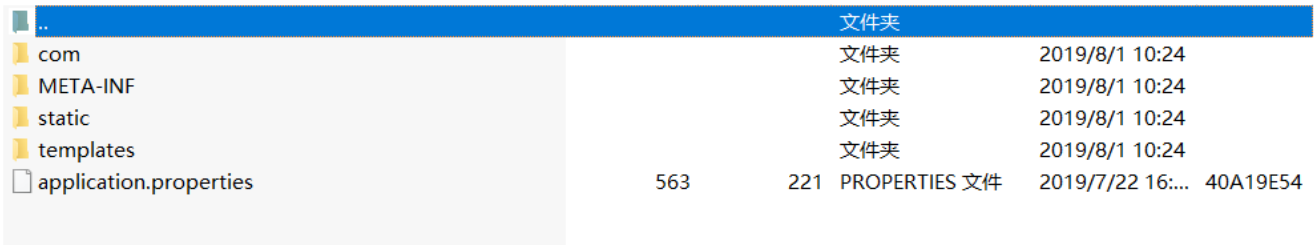




第五步：程序生产out文件夹，对应artifacts目录下有一大堆 jar 包，找到与pom.xml文件中同名的包，即程序主入口



第六步：打开主入口的包，查看是否包含 META-INF 文件夹，查看是否存在 MAMIFEST.MF 文件，再查看文件中是否指定了 Main-Class ，没有则添加上



..			文件夹		
com			文件夹	2019/8/1 10:24	
META-INF			文件夹	2019/8/1 10:24	
static			文件夹	2019/8/1 10:24	
templates			文件夹	2019/8/1 10:24	
application.properties	563	221	PROPERTIES 文件	2019/7/22 16:...	40A19E54

```
Manifest-Version: 1.0
Class-Path: snakeyaml-1.23.jar thymeleaf-3.0.11.RELEASE.jar spring-boot-
 autoconfigure-2.1.6.RELEASE.jar jboss-logging-3.3.2.Final.jar jul-to-sl
 f4j-1.7.26.jar spring-boot-starter-logging-2.1.6.RELEASE.jar spring-boo
 t-starter-tomcat-2.1.6.RELEASE.jar spring-beans-5.1.8.RELEASE.jar jacks
 on-datatype-jsr310-2.9.9.jar logback-core-1.2.3.jar spring-context-5.1.
 8.RELEASE.jar spring-boot-starter-thymeleaf-2.1.6.RELEASE.jar spring-bo
 ot-starter-2.1.6.RELEASE.jar spring-core-5.1.8.RELEASE.jar unbescape-1.
 1.6.RELEASE.jar spring-webmvc-5.1.8.RELEASE.jar jackson-annotations-2.9
 .0.jar hibernate-validator-6.0.17.Final.jar jaxb-core-2.3.0.jar jaxb-ap
 i-2.3.0.jar logback-classic-1.2.3.jar activation-1.1.1.jar tomcat-embed
 -el-9.0.21.jar spring-boot-starter-web-2.1.6.RELEASE.jar thymeleaf-spri
 ng5-3.0.11.RELEASE.jar jackson-module-parameter-names-2.9.9.jar jackson
 -datatype-jdk8-2.9.9.jar log4j-to-slf4j-2.11.2.jar validation-api-2.0.1
 .Final.jar classmate-1.4.0.jar spring-web-5.1.8.RELEASE.jar spring-boot
 -starter-json-2.1.6.RELEASE.jar tomcat-embed-core-9.0.21.jar spring-jcl
 -5.1.8.RELEASE.jar thymeleaf-extras-java8time-3.0.4.RELEASE.jar jackson
 -core-2.9.9.jar tomcat-embed-websocket-9.0.21.jar spring-expression-5.1
 .8.RELEASE.jar attoparser-2.0.5.RELEASE.jar spring-boot-2.1.6.RELEASE.j
 ar log4j-api-2.11.2.jar slf4j-api-1.7.26.jar spring-aop-5.1.8.RELEASE.j
 ar jaxb-impl-2.3.0.jar jackson-databind-2.9.9.jar javax.annotation-api-
 1.3.2.jar
Main-Class: com.LBModelProject.App
```

第七步：大功告成，到对应目录下启动主入口的 jar 包，命令：java -jar xxx.jar

(ps: 要部署到linux服务器供远程访问的话也一样，直接拷贝out文件夹到喜欢的目录，然后执行java -jar xxx.jar即可，既可以在客户端输入指定ip：端口/路径访问啦)


:: Spring Boot :: (v2.1.6.RELEASE)
2019-08-01 10:25:52.201 INFO 24500 --- [main] com.LBModelProject.App : Starting App on DESKTOP-9ATHF12 with PID 24500 (started by 97085 in G:\学校那些事\实习\LBModel\out\artifacts\LBModel_jar)
2019-08-01 10:25:52.202 INFO 24500 --- [main] com.LBModelProject.App : No active profile set, falling back to default profiles: default
2019-08-01 10:25:53.654 INFO 24500 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2019-08-01 10:25:53.695 INFO 24500 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2019-08-01 10:25:53.695 INFO 24500 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.21]
2019-08-01 10:25:53.774 INFO 24500 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2019-08-01 10:25:53.774 INFO 24500 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1540 ms
2019-08-01 10:25:53.961 INFO 24500 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2019-08-01 10:25:54.050 WARN 24500 --- [main] org.thymeleaf.templatemode.TemplateMode : [THYMELEAF][main] Template Mode 'HTML5' is deprecated. Using Template Mode 'HTML' instead.
2019-08-01 10:25:54.157 INFO 24500 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2019-08-01 10:25:54.160 INFO 24500 --- [main] com.LBModelProject.App : Started App in 2.217 seconds (JVM running for 2.456)
2019-08-01 10:26:07.476 INFO 24500 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'