P2P 项目

主讲： 郭鑫

北京动力节点教育科技有限公司

动力节点课程讲义

DONGLIJIEDIANKECHENGJIANGYI

www.bjpowernode.com

目录

第1章 互联网金融

1.1概念

互联网金融（ITFIN）是指传统金融机构与互联网企业利用互联网技术和信息通信技术实现资金融通、支付、投资和信息中介服务的新型金融业务模式。

互联网金融 ITFIN 不是互联网和金融业的简单结合，而是在实现安全、移动等网络技术水平上，被用户熟悉接受后（尤其是对电子商务的接受），自然而然为适应新的需求而产生的新模式及新业务。是传统金融行业与互联网技术相结合的新兴领域。2016 年 10 月 13 日，国务院办公厅发布《互联网金融风险专项整治工作实施方案的通知》

1.2定义

互联网金融（ITFIN）就是互联网技术和金融功能的有机结合，依托大数据和云计算在开放的互联网平台上形成的功能化金融业态及其服务体系，包括基于网络平台的金融市场体系、金融服务体系、金融组织体系、金融产品体系以及互联网金融监管体系等，并具有普惠金融、平台金融、信息金融和碎片金融等相异于传统金融的金融模式。

1.3介绍

互联网金融是传统金融机构与互联网企业（以下统称从业机构）利用互联网技术和信息通信技术实现资金融通、支付、投资和信息中介服务的新型金融业务模式。互联网与金融深度融合是大势所趋，将对金融产品、业务、组织和服务等方面产生更加深刻的影响。互联网金融对促进小微企业发展和扩大就业发挥了现有金融机构难以替代的积极作用，为大众创业、万众创新打开了大门。促进互联网金融健康发展，有利于提升金融服务质量和效率，深化金融改革，促进金融创新发展，扩大金融业对内对外开放，构建多层次金融体系。作为新生事物，互联网金融既需要市场驱动，鼓励创新，也需要政策助力，促进发展。

1.4 整体格局

当前互联网+金融格局，由传统金融机构和非金融机构组成。传统金融机构主要为传统金融业务的互联网创新以及电商化创新、APP 软件等；非金融机构则主要是指利用互联网技术进行金融运作的电商企业、（P2P）模式的网络借贷平台，众筹模式的网络投资平台，挖财类（模式）的手机理财 APP(理财宝类)，以及第三方支付平台等。

1.5 发展模式

1.5.1 众筹

众筹大意为大众筹资或群众筹资，是指用团购预购的形式，向网友募集项目资金的模式。众筹的本意是利用互联网和 SNS（社交网络服务）传播的特性，让创业企业、艺术家或个人对公众展示他们的创意及项目，争取大家的关注和支持，进而获得所需要的资金援助。众筹平台的运作模式大同小异——需要资金的个人或团队将项目策划交给众筹平台，经过相关审核后，便可以在平台的网站上建立属于自己的页面，用来向公众介绍项目情况。

1.5.2 P2P 网贷

P2P(Peer-to-Peerlending)，即点对点信贷。P2P 网贷是指通过第三方互联网平台进行资金借、贷双方的匹配，需要借贷的人群可以通过网站平台寻找到有出借能力并且愿意基于一定条件出借的人群，帮助贷款人通过和其他贷款人一起分担一笔借款额度来分散风险，也帮助借款人在充分比较的信息中选择有吸引力的利率条件。

两种运营模式，第一是纯线上模式，其特点是资金借贷活动都通过线上进行，不结合线下的审核。通常这些企业采取的审核借款人资质的措施有通过视频认证、查看银行流水账单、身份认证等。第二种是线上线下结合的模式，借款人在线上提交借款申请后，平台通过所在城市的代理商采取入户调查的方式审核借款人的资信、还款能力等情况。

1.5.3 第三方支付

第三方支付（Third-PartyPayment）狭义上是指具备一定实力和信誉保障的非银行机构，

借助通信、计算机和信息安全技术，采用与各大银行签约的方式，在用户与银行支付结算系统间建立连接的电子支付模式。

根据央行 2010 年在《非金融机构支付服务管理办法》中给出的非金融机构支付服务的定义，从广义上讲第三方支付是指非金融机构作为收、付款人的支付中介所提供的网络支付、预付卡、银行卡收单以及中国人民银行确定的其他支付服务。第三方支付已不仅仅局限于最初的互联网支付，而是成为线上线下全面覆盖，应用场景更为丰富的综合支付工具。

1.5.4 数字货币

除去蓬勃发展的第三方支付、P2P 贷款模式、小贷模式、众筹融资、余额宝模式等形式，以比特币为代表的互联网货币也开始露出自己的獠牙

以比特币等数字货币为代表的互联网货币爆发，从某种意义上来说，比其他任何互联网金融形式都更具颠覆性。在 2013 年 8 月 19 日，德国政府正式承认比特币的合法“货币”地位，比特币可用于缴税和其他合法用途，德国也成为全球首个认可比特币的国家。这意味着比特币开始逐渐“洗白”，从极客的玩物，走入大众的视线。也许，它能够催生出真正的互联网金融帝国。

比特币炒得火热，也跌得惨烈。无论怎样，这场似乎曾经离我们很遥远的互联网淘金盛宴已经慢慢走进我们的视线，它让人们看到了互联网金融最终极的形态就是互联网货币。所有的互联网金融只是对现有的商业银行、证券公司提出挑战，将来发展到互联网货币的形态就是对央行的挑战。也许比特币会颠覆传统金融成长为首个全球货币，也许它会最终走向崩盘，不管怎样，可以肯定的是，比特币会给人类留下一笔永恒的遗产。

1.5.5 大数据金融

大数据金融是指集合海量非结构化数据，通过对其进行实时分析，可以为互联网金融机构提供客户全方位信息，通过分析和挖掘客户的交易和消费信息掌握客户的消费习惯，并准确预测客户行为，使金融机构和金融服务平台在营销和风险控制方面有的放矢 。

基于大数据的金融服务平台主要指拥有海量数据的电子商务企业开展的金融服务。大数据的关键是从大量数据中快速获取有用信息的能力，或者是从大数据资产中快速变现利用的能力。因此，大数据的信息处理往往以云计算为基础。

1.5.6 信息化金融机构

所谓信息化金融机构，是指通过采用信息技术，对传统运营流程进行改造或重构，实现经营、管理全面电子化的银行、证券和保险等金融机构。金融信息化是金融业发展趋势之一，而信息化金融机构则是金融创新的产物。

从金融整个行业来看，银行的信息化建设一直处于业内领先水平，不仅具有国际领先的金融信息技术平台，建成了由自助银行、电话银行、手机银行和网上银行构成的电子银行立体服务体系，而且以信息化的大手笔——数据集中工程在业内独领风骚，其除了基于互联网的创新金融服务之外，还形成了“门户”“网银、金融产品超市、电商”的一拖三的金融电商创新服务模式。

1.5.7 金融门户

互联网金融门户（ITFIN）是指利用互联网进行金融产品的销售以及为金融产品销售提供第三方服务的平台。它的核心就是“搜索比价”的模式，采用金融产品垂直比价的方式，将各家金融机构的产品放在平台上，用户通过对比挑选合适的金融产品。

互联网金融门户多元化创新发展，形成了提供高端理财投资服务和理财产品的第三方理财机构，提供保险产品咨询、比价、购买服务的保险门户网站等。这种模式不存在太多政策风险，因为其平台既不负责金融产品的实际销售，也不承担任何不良的风险，同时资金也完全不通过中间平台。

1.5.8 主要特点

（1） 成本低

互联网金融模式下，资金供求双方可以通过网络平台自行完成信息甄别、匹配、定价和交易，无传统中介、无交易成本、无垄断利润。一方面，金融机构可以避免开设营业网点的资金投入和运营成本；另一方面，消费者可以在开放透明的平台上快速找到适合自己的金融产品，削弱了信息不对称程度，更省时省力。

（2） 效率高

互联网金融业务主要由计算机处理，操作流程完全标准化，客户不需要排队等候，业务处理速度更快，用户体验更好。如阿里小贷依托电商积累的信用数据库，经过数据挖掘和分析，引入风险分析和资信调查模型，商户从申请贷款到发放只需要几秒钟，日均可以完成贷款 1 万笔，成为真正的“信贷工厂”。

（3） 覆盖广

互联网金融模式下，客户能够突破时间和地域的约束，在互联网上寻找需要的金融资源，金融服务更直接，客户基础更广泛。此外，互联网金融的客户以小微企业为主，覆盖了部分传统金融业的金融服务盲区，有利于提升资源配置效率，促进实体经济发展。

（4） 发展快

依托于大数据和电子商务的发展，互联网金融得到了快速增长。以余额宝为例，余额宝上线 18 天，累计用户数达到 250 多万，累计转入资金达到 66 亿元。据报道，余额宝规模 500 亿元，成为规模最大的公募基金。

（5） 管理弱

一是风控弱。互联网金融还没有接入人民银行征信系统，也不存在信用信息共享机制，不具备类似银行的风控、合规和清收机制，容易发生各类风险问题，已有众贷网、网赢天下等 P2P 网贷平台宣布破产或停止服务。二是监管弱。互联网金融在中国处于起步阶段，还没有监管和法律约束，缺乏准入门槛和行业规范，整个行业面临诸多政策和法律风险。

（6） 风险大

一是信用风险大。现阶段中国信用体系尚不完善，互联网金融的相关法律还有待配套，互联网金融违约成本较低，容易诱发恶意骗贷、卷款跑路等风险问题。特别是 P2P 网贷平台由于准入门槛低和缺乏监管，成为不法分子从事非法集资和诈骗等犯罪活动的温床。去年以

来，淘金贷、优易网、安泰卓越等 P2P 网贷平台先后曝出“跑路”事件。

二是网络安全风险大。中国互联网安全问题突出，网络金融犯罪问题不容忽视。一旦遭遇黑客攻击，互联网金融的正常运作会受到影响，危及消费者的资金安全和个人信息安全。

第2章 互联网金融—P2P 项目

2.1 业务模式

2.2 P2P 网贷平台构成

2.2.1 理财端

理财端通常是围绕出借人业务展开的。理财端展现方式：PC 电脑、H5 页面、APP 应用

2.2.2 借款端

借款端展现方式：PC 电脑

2.3 项目基本开发流程

1) 公司领导层做出决策；
2) 产品经理根据领导决策勾勒产品需求原型；
3) 产品+设计+技术+测试 需求评审讨论；
4) 评审讨论调整后进入设计阶段，视觉设计+页面设计
5) 技术团队技术架构+技术选型+存储+服务器+运维等

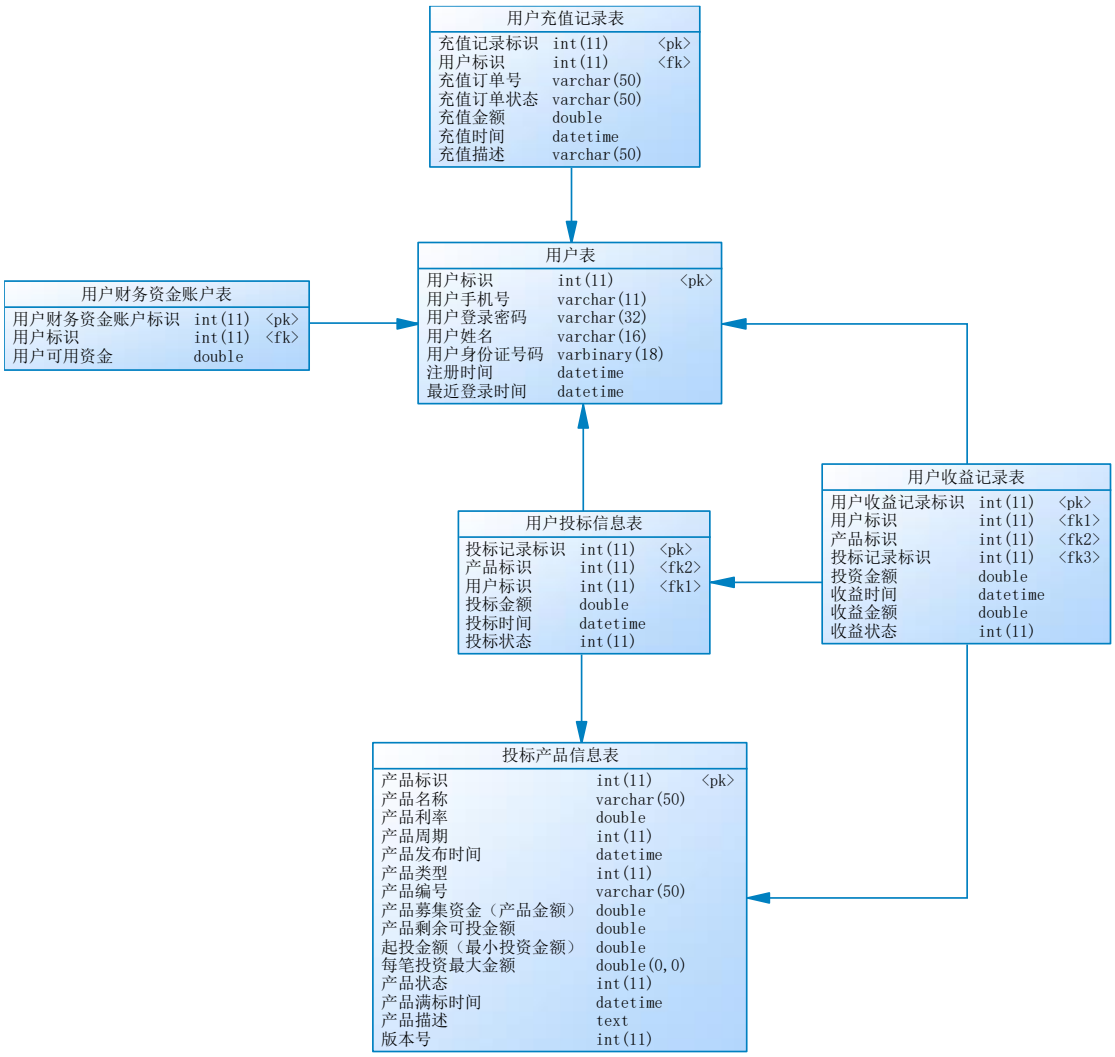
2.4 动力金融网技术选型及开发工具

- 前端：jQuery + Thymeleaf
- 后端：SpringBoot + SpringMVC + Spring + Dubbo + mybatis
- 相关组件：HttpComponents、Apache commons、Druid、spring-data-redis
- 服务器端：Tomcat、Redis、Zookeeper、MySQL
- 项目依赖管理：Maven
- 项目开发工具：IDEA

第3章 搭建 P2P 项目架构

3.1P2P 项目数据库

3.1.1 数据库表结构



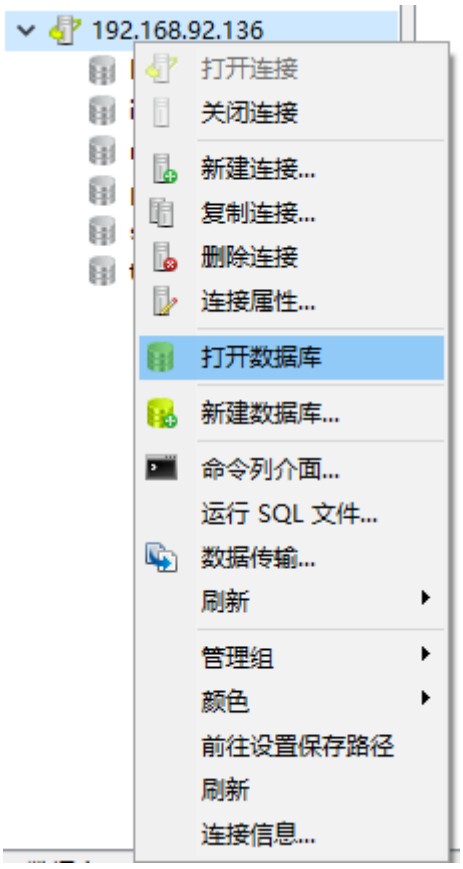
3.1.2 初始化数据库

（1） 开启 linux 服务器 MySQL 服务

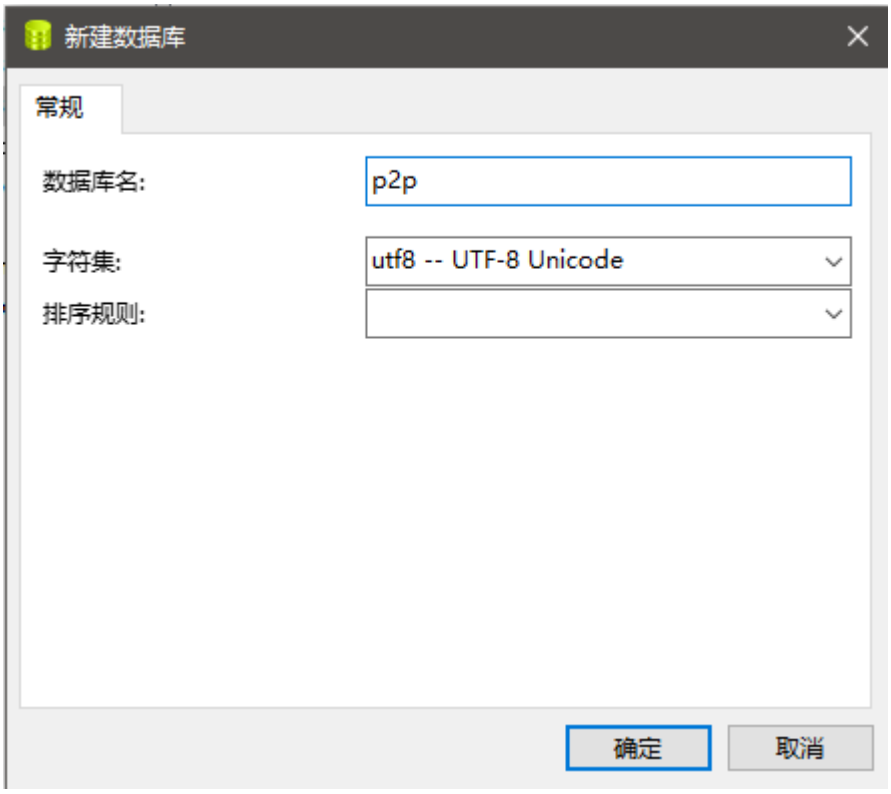
```
[root@localhost bin]# ./mysqld_safe &
[1] 2857
[root@localhost bin]# Logging to '/usr/local/mysql-5.7.18/data/localhost.localdomain.err'.
2019-08-09T08:51:01.493943Z mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql-5.7.18/data
```

（2） 使用 Navicat 客户端创建数据库：p2p

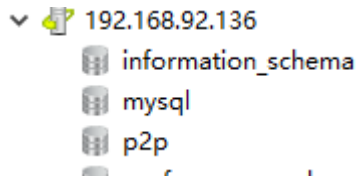
A、 选中服务器，鼠标右击->打开数据库



B、 输入数据名及选择字符集

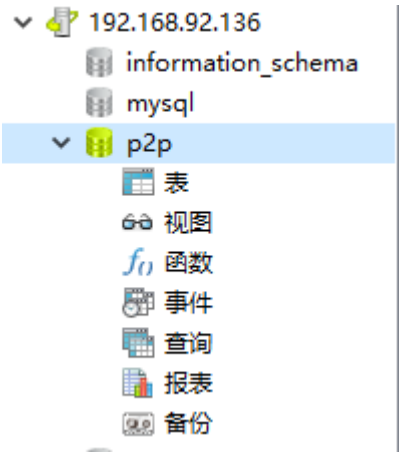


C、 创建成功

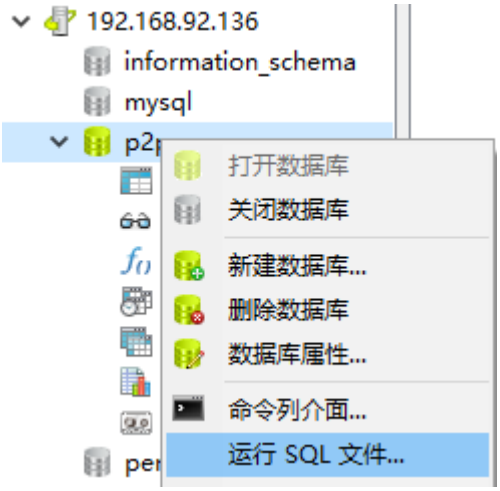


（3） 初始化 p2p 数据库

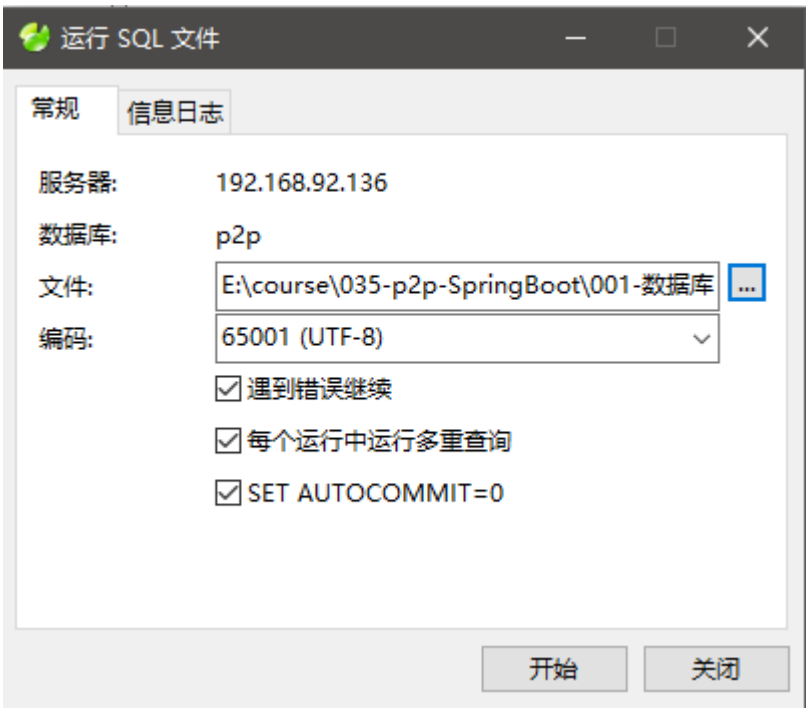
D、 选中 p2p 数据库，双击打开



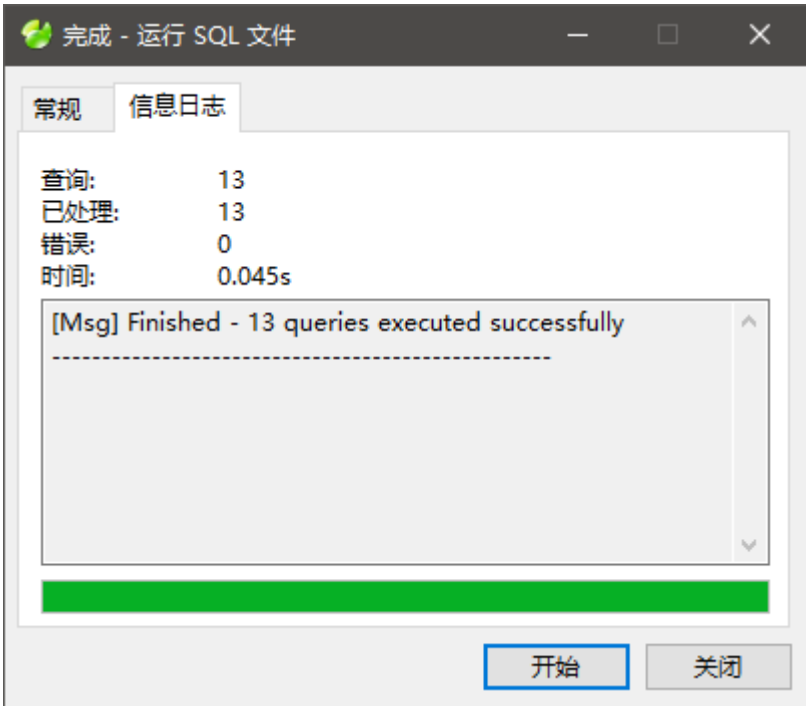
E、选中 p2p 数据库， 鼠标右键->运行 SQL 文件



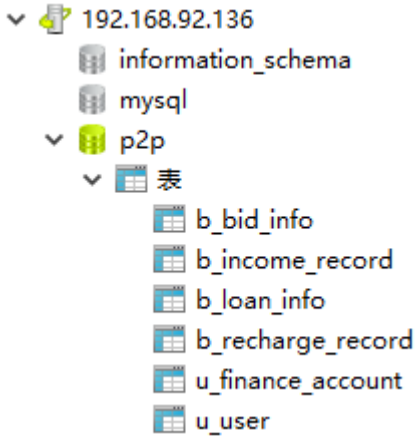
F、选择 p2p 数据库脚本文件，点击开始



G、 关闭窗口

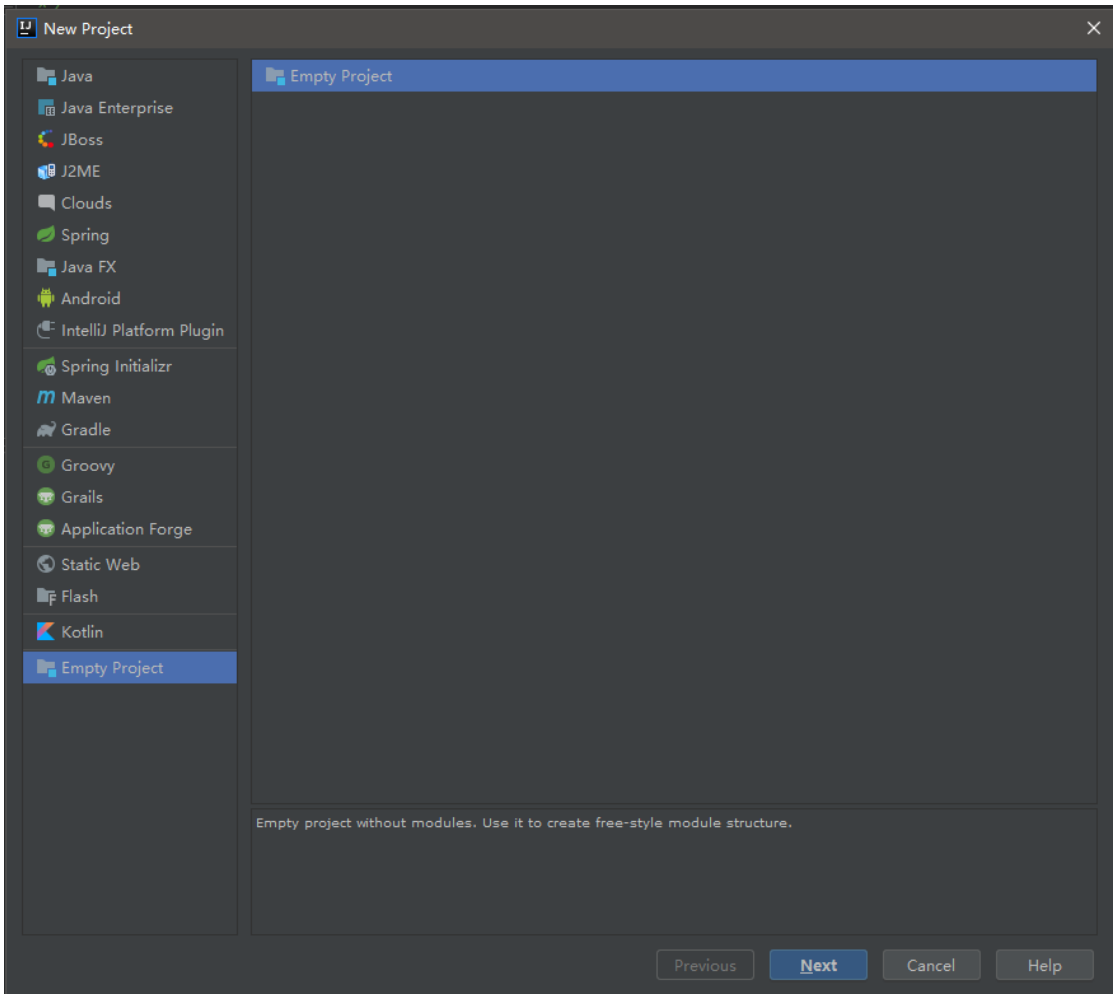


H、 刷新 p2p 数据库

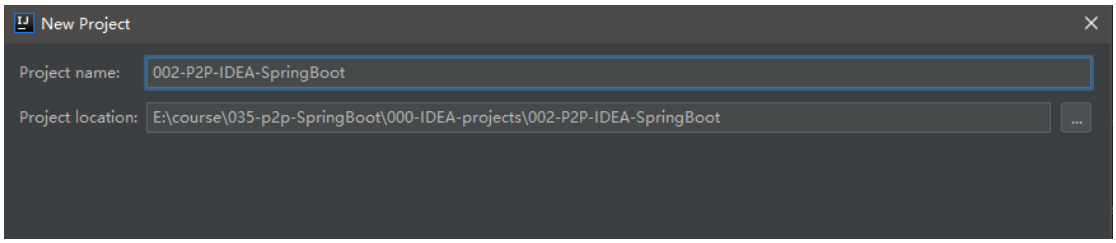


3.2 创建项目工作空间

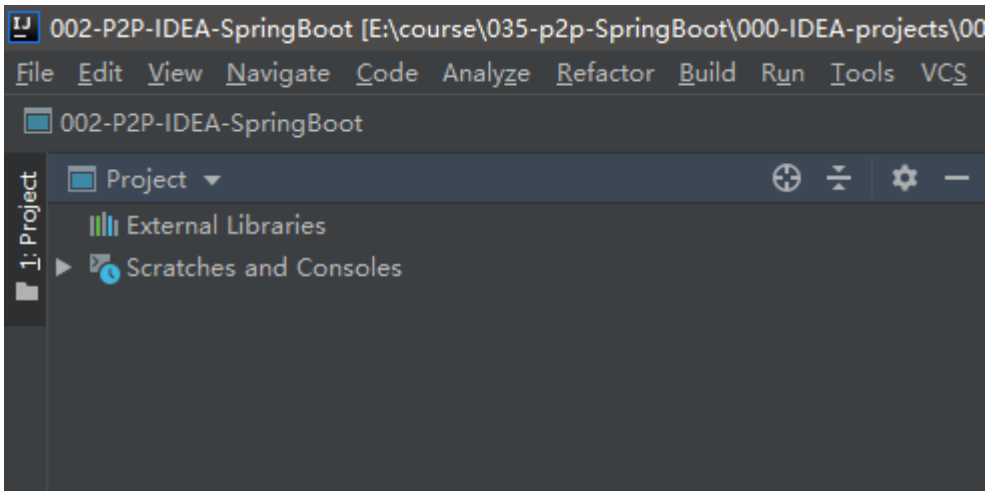
3.2.3 选择创建 Empty Project



3.2.4 设置名称及位置

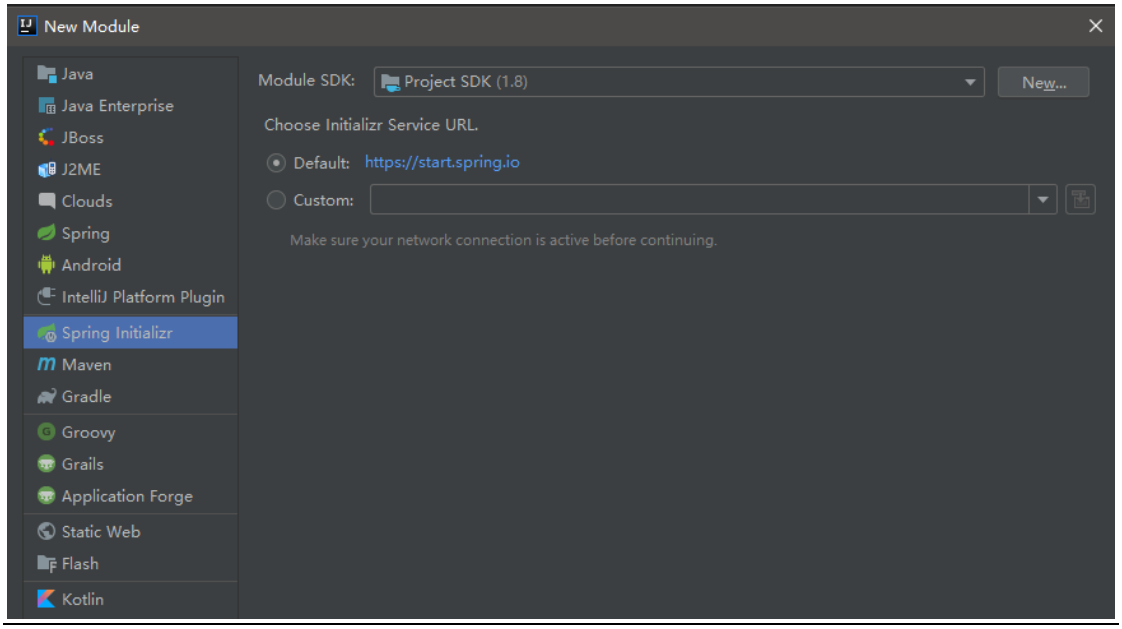


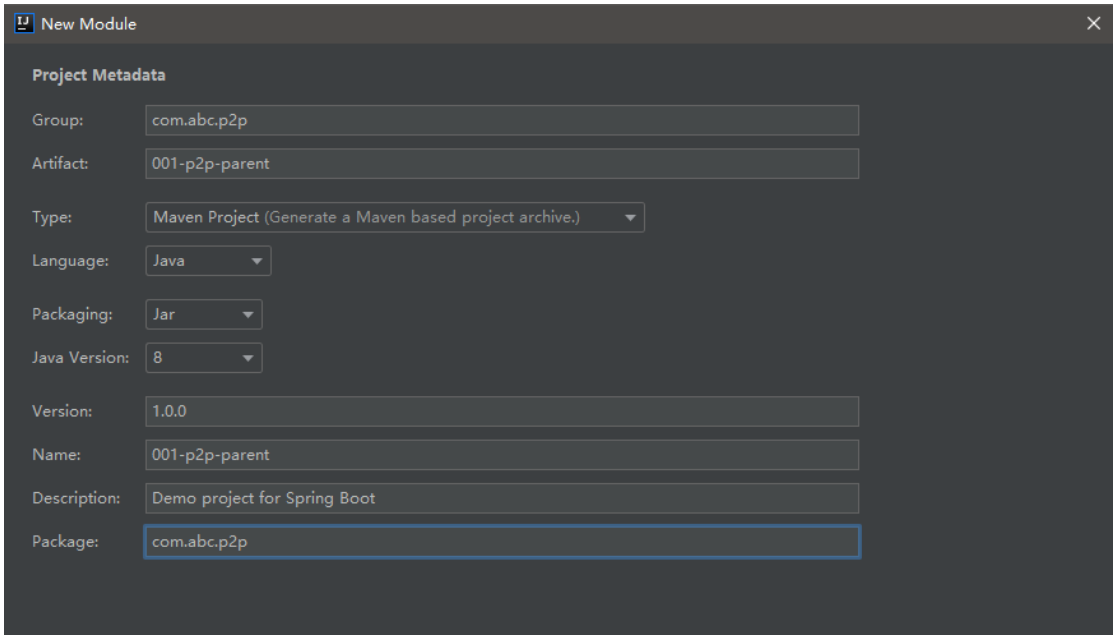
3.2.5 查看工作空间

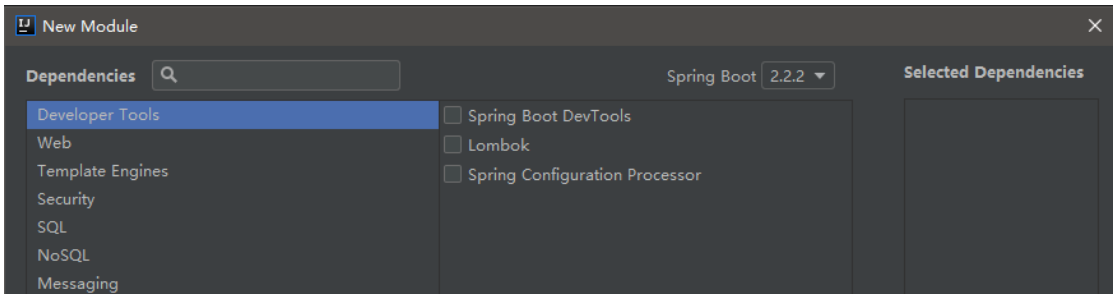


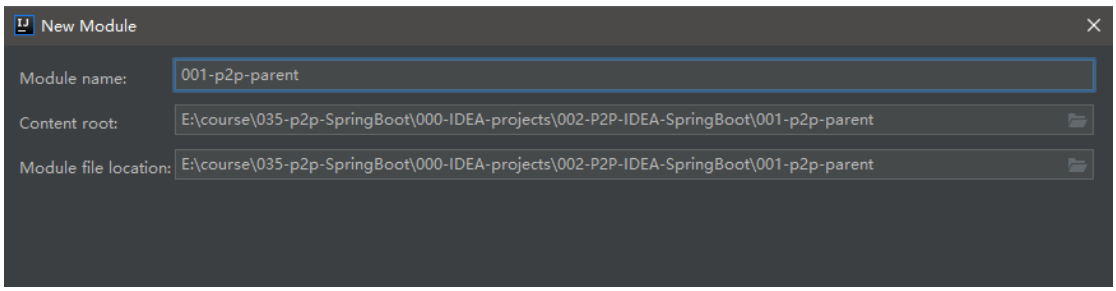
3.3 创建整体项目父工程

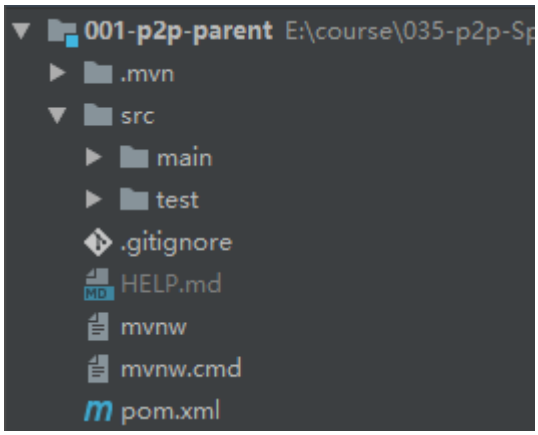
3.3.6 项目名称：001-p2p-parent



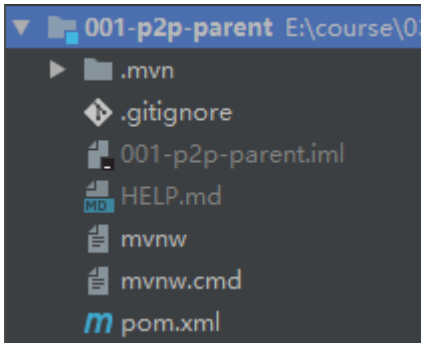

New Module
Project Metadata
Group: com.abc.p2p
Artifact: 001-p2p-parent
Type: Maven Project (Generate a Maven based project archive.)
Language: Java
Packaging: Jar
Java Version: 8
Version: 1.0.0
Name: 001-p2p-parent
Description: Demo project for Spring Boot
Package: com.abc.p2p


New Module
Dependencies
Spring Boot 2.2.2
Selected Dependencies
Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
Spring Boot DevTools
Lombok
Spring Configuration Processor


New Module
Module name: 001-p2p-parent
Content root: E:\course\035-p2p-SpringBoot\000-IDEA-projects\002-P2P-IDEA-SpringBoot\001-p2p-parent
Module file location: E:\course\035-p2p-SpringBoot\000-IDEA-projects\002-P2P-IDEA-SpringBoot\001-p2p-parent


001-p2p-parent E:\course\035-p2p-Sp
.mvn
src
main
test
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml

3.3.7 删除 src 目录



3.3.8 将 pom 文件中 packaging 标签设置为 pom

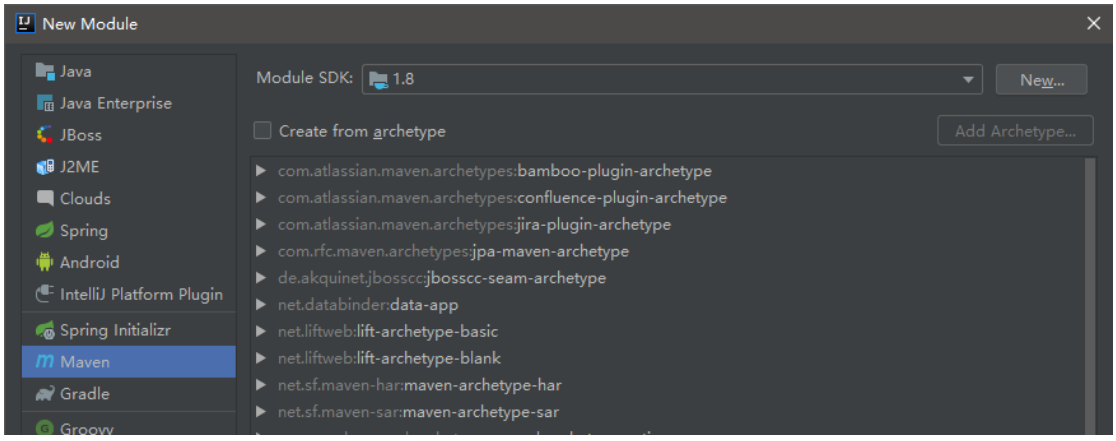
```
<!--父工程的 packaging 必须设置为 pom-->

<packaging>pom</packaging>
```

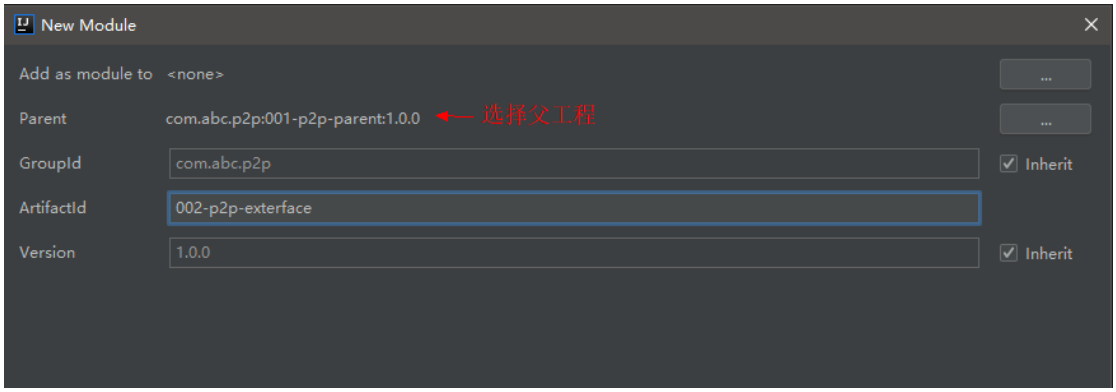
3.3.9 添加父工程要管理的依赖

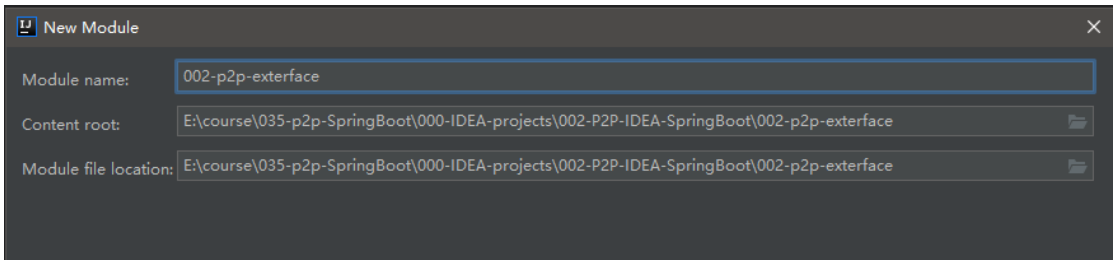
3.4 创建接口工程

普通 Maven 结构 Java 工程

3.4.10 项目名称：002-p2p-exterface



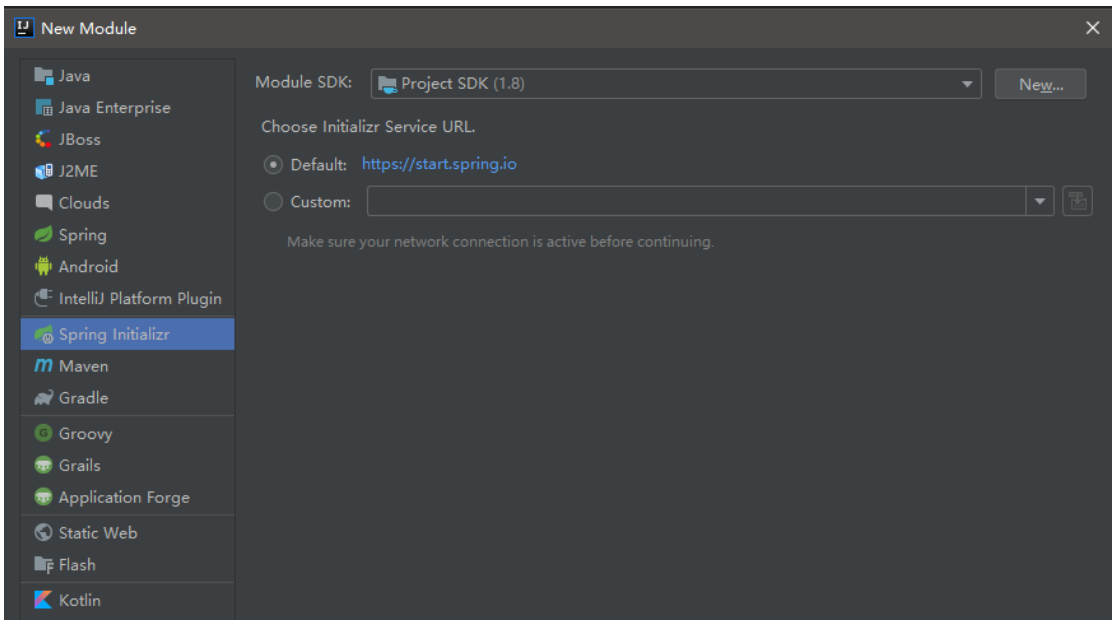


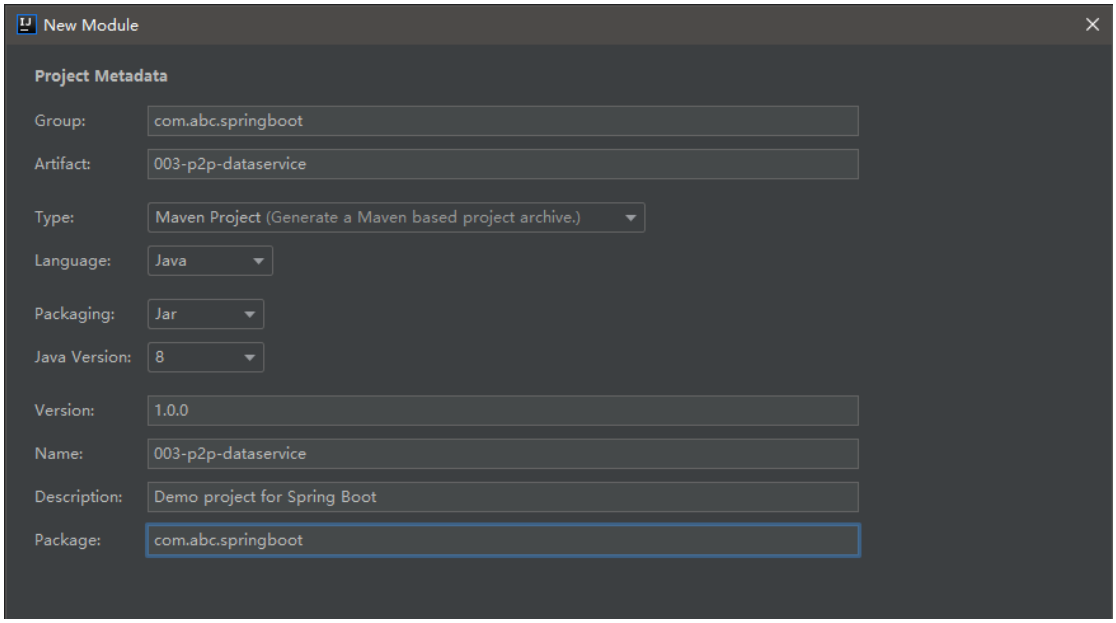


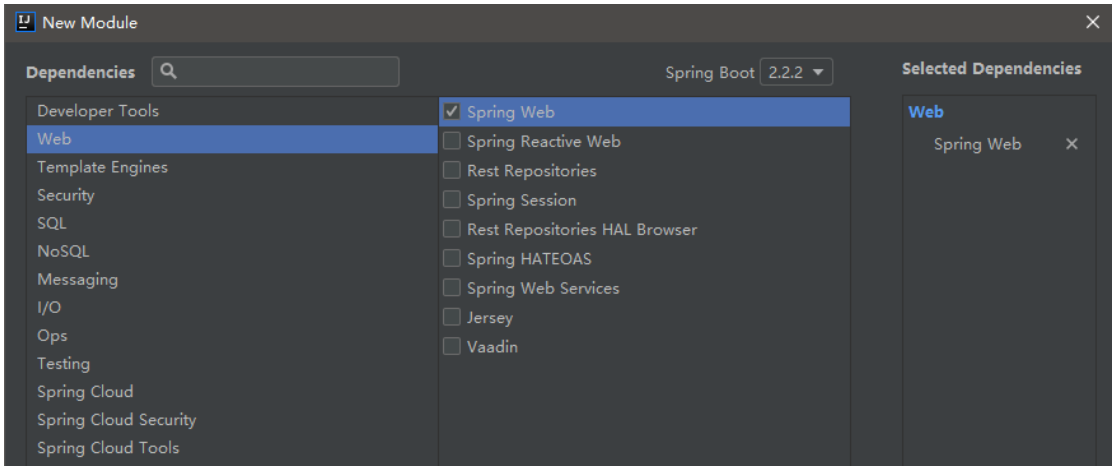
3.5项目服务提供者工程

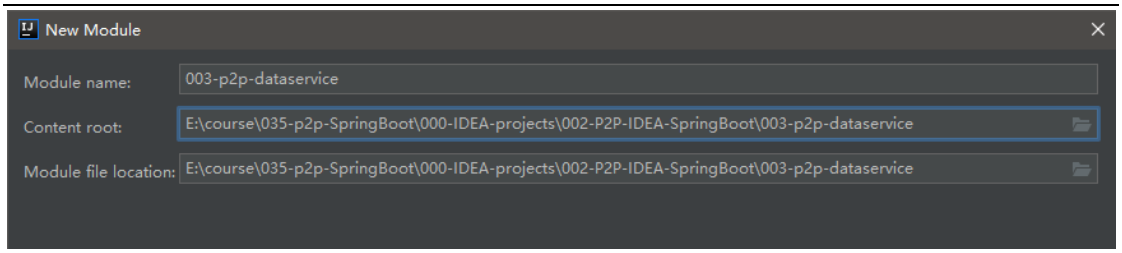
SpringBoot 框架 web 项目

3.5.11 项目名称：003-p2p-dataservice









3.5.12 修改父工程为 001-p2p-parent

```
<parent>
    <artifactId>001-p2p-parent</artifactId>
    <groupId>com.abc.p2p</groupId>
    <version>1.0.0</version>
    <relativePath>../001-p2p-parent/pom.xml</relativePath>
</parent>

<artifactId>003-p2p-dataservice</artifactId>
```

3.5.13 添加依赖

```
<!--SpringBoot 框架 web 项目起步依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!--MyBatis 集成 SpringBoot 框架-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
</dependency>

<!--MySQL 数据库驱动-->
<dependency>
    <groupId>mysql</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
</dependency>

<!--Dubbo 集成 SpringBoot-->
<dependency>
    <groupId>com.alibaba.spring.boot</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
</dependency>

<!--zookeeper 注册中心-->
<dependency>
    <groupId>com.101tec</groupId>
    <artifactId>zkclient</artifactId>
</dependency>

<!--SpringBoot 集成 redis-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>

<!--Apache commons-->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
</dependency>

<!--接口工程-->
<dependency>
```

```
    <groupId>com.abc.p2p</groupId>
    <artifactId>002-p2p-exterface</artifactId>
    <version>1.0.0</version>
</dependency>

<!--公共工程-->
<dependency>
    <groupId>com.abc.p2p</groupId>
    <artifactId>004-p2p-common</artifactId>
    <version>1.0.0</version>
</dependency>
```

3.5.14 添加日志文件

将日志文件 logback-spring.xml 添加到 resources 根目录下

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

<!-- 日志级别从低到高分为 TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为 WARN，则低于
WARN 的信息都不会输出 -->
<!-- scan:当此属性设置为 true 时，配置文件如果发生改变，将会被重新加载，默认值为 true -->
<!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当 scan
为 true 时，此属性生效。默认的时间间隔为 1 分钟。 -->
<!-- debug:当此属性设置为 true 时，将打印出 logback 内部日志信息，实时查看 logback 运行状态。默认值为
false。通常不打印 -->
<configuration  scan="true" scanPeriod="10 seconds">

    <!--输出到控制台-->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <!--此日志 appender 是为开发使用，只配置最低级别，控制台输出的日志级别是大于或等于此级别的日志
信息-->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
```

```
        <level>debug</level>
      </filter>
      <encoder>
        <Pattern>%date [%-5p] [%thread] %logger{60} [%file : %line] %msg%n</Pattern>
        <!-- 设置字符集 -->
        <charset>UTF-8</charset>
      </encoder>
   </appender>

   <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
      <!--<File>/home/log/stdout.log</File>-->
      <File>D:/log/p2p-dataservice.log</File>
      <encoder>
        <pattern>%date [%-5p] %thread %logger{60} [%file : %line] %msg%n</pattern>
      </encoder>
      <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- 添加.gz 历史日志会启用压缩 大大缩小日志文件所占空间 -->

<!--<fileNamePattern>/home/log/p2p-dataservice.log.%d{yyyy-MM-dd}.log</fileNamePattern
>-->

<fileNamePattern>D:/log/p2p-dataservice.log.%d{yyyy-MM-dd}.log</fileNamePattern>
        <maxHistory>30</maxHistory><!--  保留 30 天日志 -->
      </rollingPolicy>
   </appender>

   <logger name="com.abc.p2p.mapper" level="DEBUG" />

   <root level="INFO">
```

```
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
    </root>
</configuration>
```

3.5.15 设置核心配置文件

（4） application.properties

```
#设置服务端口号
server.port=8081
#设置上下文根
server.servlet.context-path=/

#配置数据源
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/p2p?useUnicode=true
&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true&useLegacy
DatetimeCode=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=123456

#配置 Dubbo 的服务提供者
spring.application.name=003-p2p-dataservice
#设置工程为服务提供者
spring.dubbo.server=true
#设置 dubbo 注册中心
spring.dubbo.registry=zookeeper://localhost:2181
```

```
#配置 Redis 缓存
spring.redis.host=localhost
spring.redis.timeout=15000
spring.redis.port=6379
spring.redis.password=123456
```

（5） application.yml

```
server:
  port: 8081
  servlet:
    context-path: /

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/p2p?useUnicode=true&characterEncoding=UTF
-8&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&ser
verTimezone=UTC
    username: root
    password: 123456
  application:
    name: 003-p2p-dataservice
  dubbo:
    server: true
  registry: zookeeper://localhost:2181
  redis:
    host: localhost
```

```
    timeout: 15000
    port: 6379
    password: 123456
```

3.5.16 MyBatis 逆向工程

（6） 添加逆向工程插件

```
<!--mybatis 代码自动生成插件-->
<plugin>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.3.7</version>
    <configuration>
        <!--配置文件的位置-->
        <configurationFile>GeneratorMapper.xml</configurationFile>
        <verbose>true</verbose>
        <overwrite>true</overwrite>
    </configuration>
</plugin>
```

（7） 将逆向工程文件存放到项目根目录下

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
      PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
      "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<generatorConfiguration>
```

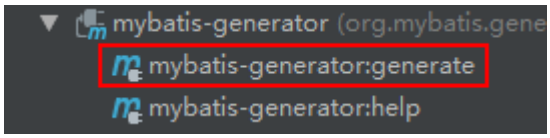
```
    <!-- 指定连接数据库的 JDBC 驱动包所在位置，指定到你本机的完整路径 -->
    <classPathEntry location="E:\mysql-connector-java-5.1.38.jar"/>

    <!-- 配置 table 表信息内容体，targetRuntime 指定采用 MyBatis3 的版本 -->
    <context id="tables" targetRuntime="MyBatis3">

        <!-- 抑制生成注释，由于生成的注释都是英文的，可以不让它生成 -->
        <commentGenerator>
            <property name="suppressAllComments" value="true" />
        </commentGenerator>

        <!-- 配置数据库连接信息 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
                        connectionURL="jdbc:mysql://192.168.92.134:3306/p2p"
                        userId="root"
                        password="123456">
        </jdbcConnection>

        <!-- 生成 model 类，targetPackage 指定 model 类的包名， targetProject 指定生成的 model 放在
eclipse 的哪个工程下面-->
        <javaModelGenerator targetPackage="com.abc.p2p.model"

targetProject="E:\course\035-p2p-SpringBoot\000-IDEA-projects\002-P2P-IDEA-SpringBoot\
002-p2p-exterface\src\main\java">
            <property name="enableSubPackages" value="false" />
            <property name="trimStrings" value="false" />
        </javaModelGenerator>
```

```
        <!-- 生成 MyBatis 的 Mapper.xml 文件，targetPackage 指定 mapper.xml 文件的包名，
targetProject 指定生成的 mapper.xml 放在 eclipse 的哪个工程下面 -->
        <sqlMapGenerator targetPackage="com.abc.p2p.mapper"
targetProject="E:\course\035-p2p-SpringBoot\000-IDEA-projects\002-P2P-IDEA-SpringBoot\
003-p2p-dataservice\src\main\java">
            <property name="enableSubPackages" value="false" />
        </sqlMapGenerator>

        <!-- 生成 MyBatis 的 Mapper 接口类文件,targetPackage 指定 Mapper 接口类的包名，
targetProject 指定生成的 Mapper 接口放在 eclipse 的哪个工程下面 -->
        <javaClientGenerator type="XMLMAPPER" targetPackage="com.abc.p2p.mapper"
targetProject="E:\course\035-p2p-SpringBoot\000-IDEA-projects\002-P2P-IDEA-SpringBoot\
003-p2p-dataservice\src\main\java">
            <property name="enableSubPackages" value="false" />
        </javaClientGenerator>

        <!-- 数据库表名及对应的 Java 模型类名 -->
        <table tableName="b_loan_info" domainObjectName="LoanInfo"
                enableCountByExample="false"
                enableUpdateByExample="false"
                enableDeleteByExample="false"
                enableSelectByExample="false"
                selectByExampleQueryId="false"/>

        <table tableName="b_bid_info" domainObjectName="BidInfo"
                enableCountByExample="false"
                enableUpdateByExample="false"
                enableDeleteByExample="false"
                enableSelectByExample="false"
```

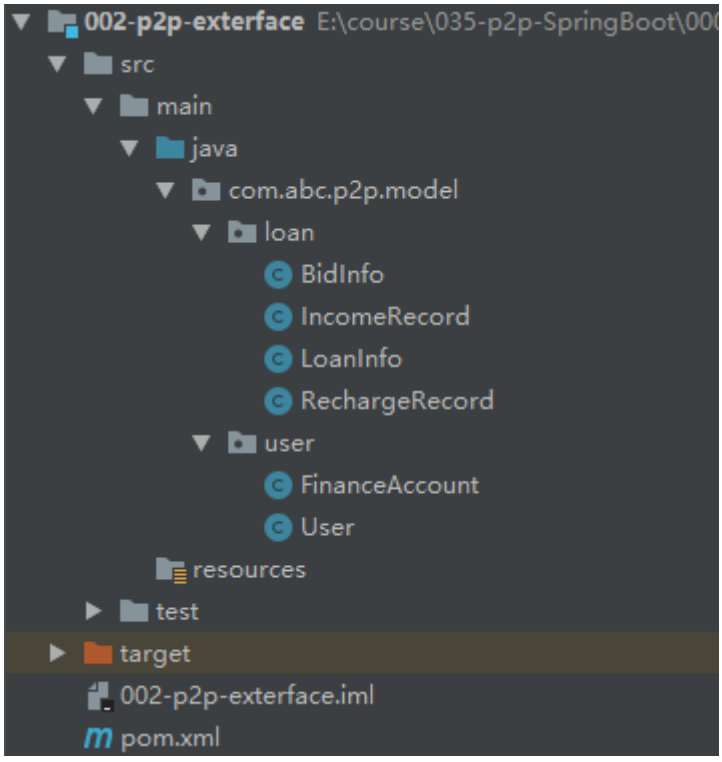
```
        selectByExampleQueryId="false"/>



    <table tableName="b_income_record" domainObjectName="IncomeRecord"
        enableCountByExample="false"
        enableUpdateByExample="false"
        enableDeleteByExample="false"
        enableSelectByExample="false"
        selectByExampleQueryId="false"/>



    <table tableName="b_recharge_record" domainObjectName="RechargeRecord"
        enableCountByExample="false"
        enableUpdateByExample="false"
        enableDeleteByExample="false"
        enableSelectByExample="false"
        selectByExampleQueryId="false"/>



    <table tableName="u_user" domainObjectName="User"
        enableCountByExample="false"
        enableUpdateByExample="false"
        enableDeleteByExample="false"
        enableSelectByExample="false"
        selectByExampleQueryId="false"/>



    <table tableName="u_finance_account" domainObjectName="FinanceAccount"
        enableCountByExample="false"
```

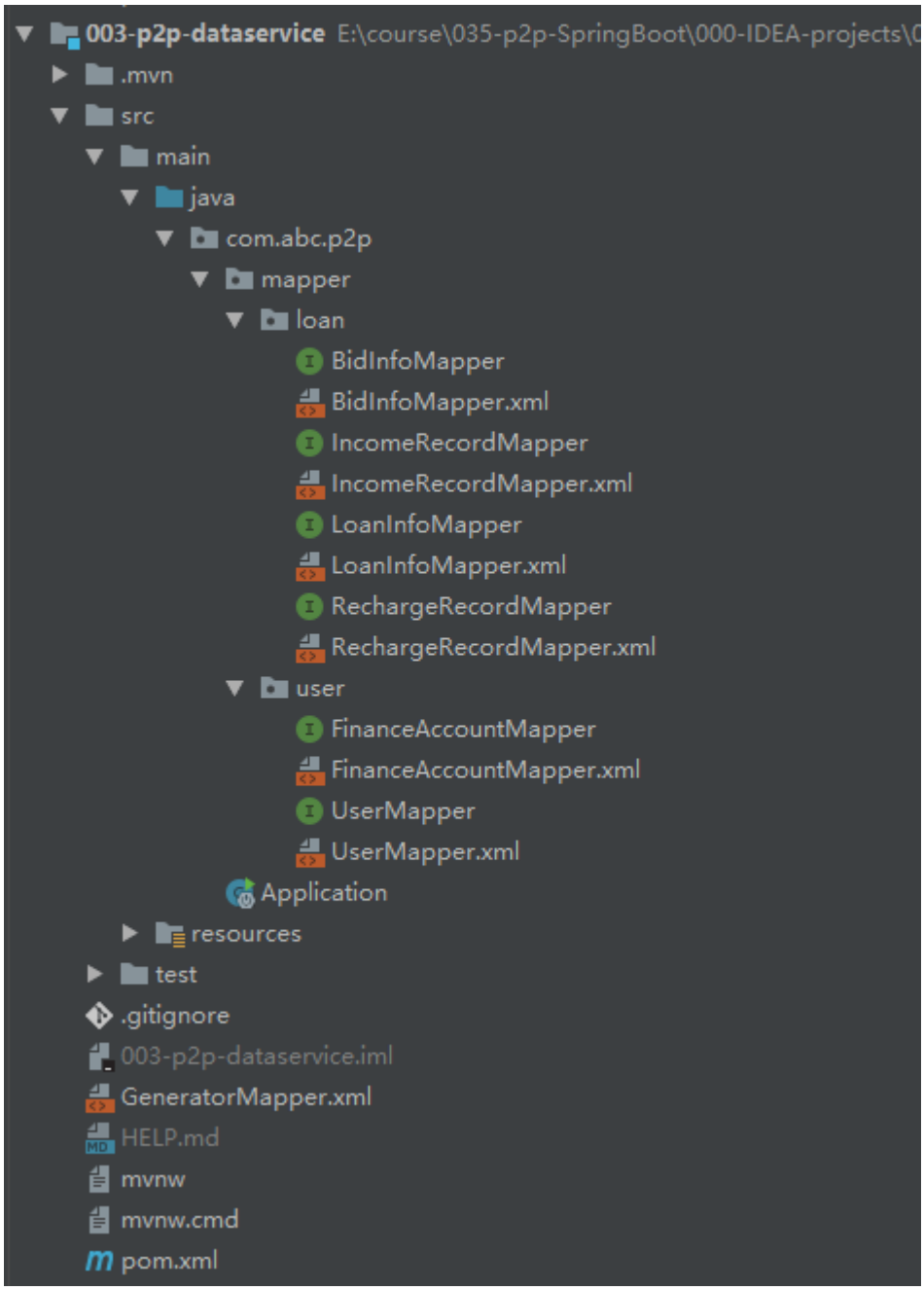
```
            enableUpdateByExample="false"
            enableDeleteByExample="false"
            enableSelectByExample="false"
            selectByExampleQueryId="false"/>
    </context>

</generatorConfiguration>
```

（8） 双击生成



（9） 项目包结构再次细化





3.5.17 项目启动类

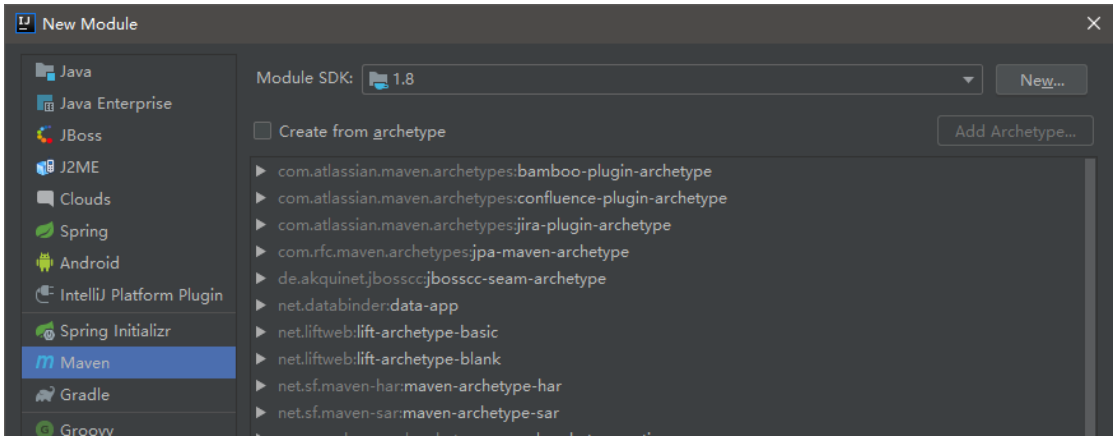
```
@SpringBootApplication

@EnableDubboConfiguration   //启动 Dubbo 配置

@MapperScan(basePackages = "com.abc.p2p.mapper")

public class Application {



    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);
```

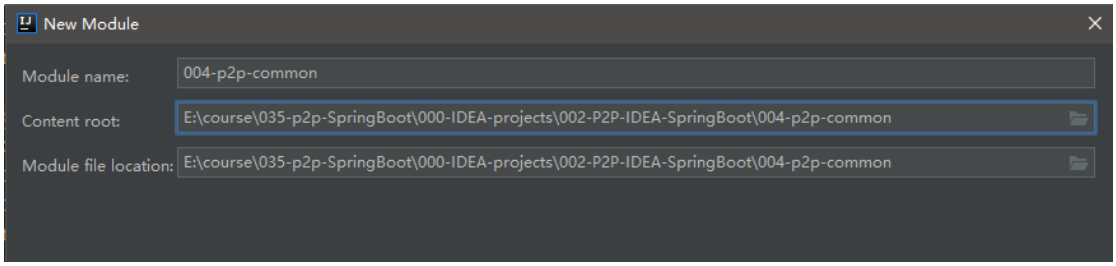
```
    }
}
```

3.6 公共项目

普通 Maven 结构 Java 工程，用来存放常用工具类、常量类等公共项目内容。

3.6.18 项目名称：004-p2p-common

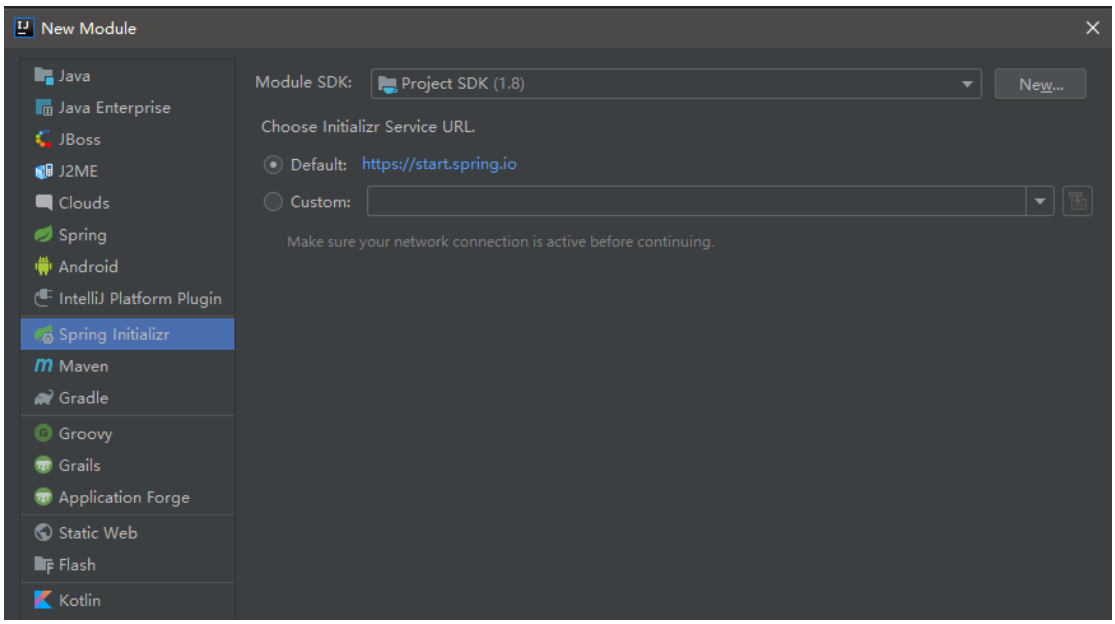


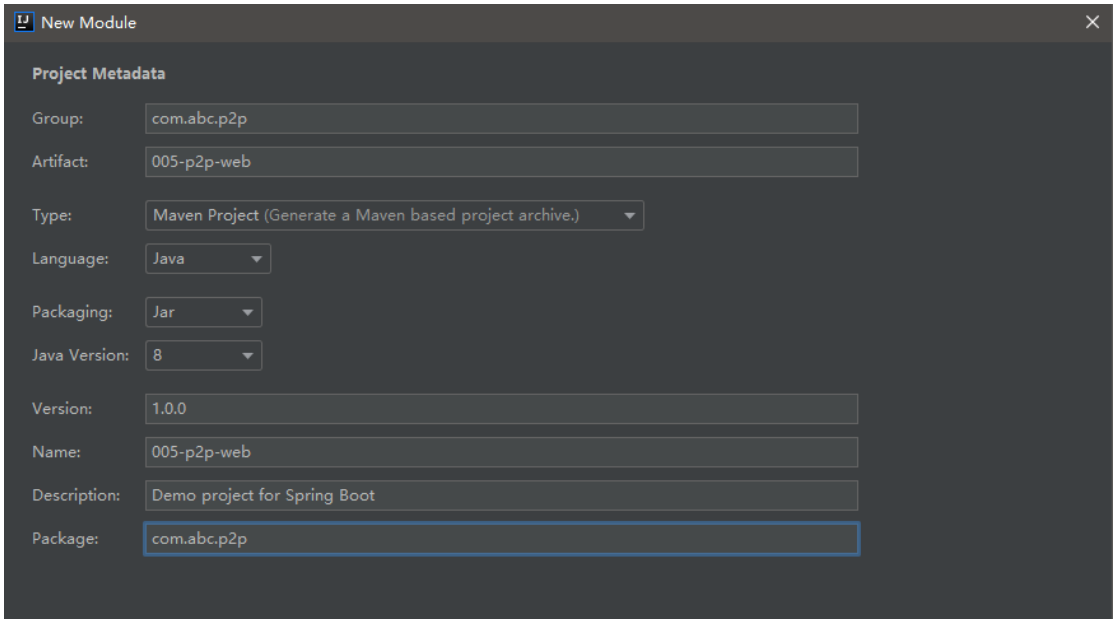


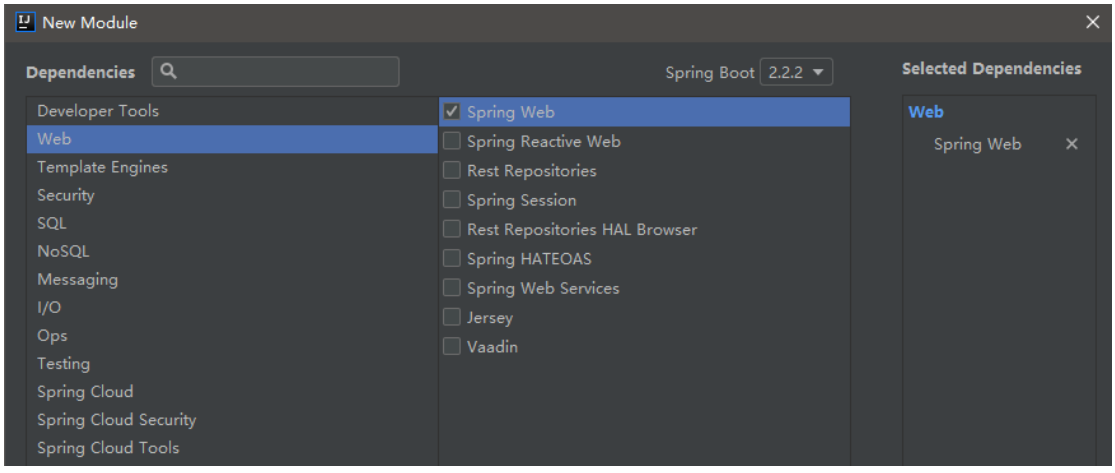
3.7 项目服务消费者工程

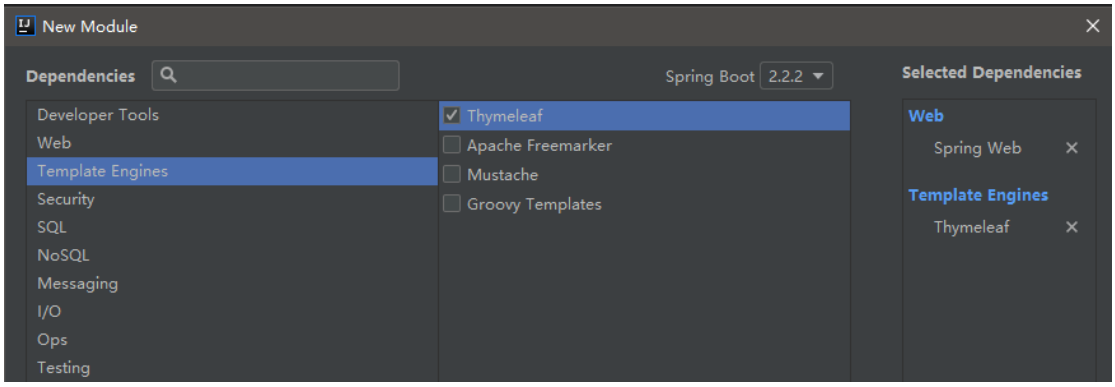
SpringBoot 框架 web 项目，并集成 Thymeleaf 模版

3.7.19 项目名称：005-p2p-web









3.7.20 修改父工程为 001-p2p-parent

```
<parent>
    <artifactId>001-p2p-parent</artifactId>
    <groupId>com.abc.p2p</groupId>
    <version>1.0.0</version>
    <relativePath>../001-p2p-parent/pom.xml</relativePath>
</parent>

<artifactId>005-p2p-web</artifactId>
```

3.7.21 添加依赖

```
<!--SprinBoot 框架集成 Thymeleaf 模版起步依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<!--SpringBoot 框架 web 项目起步依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
<!--Dubbo 集成 SpringBoot 依赖-->
<dependency>
    <groupId>com.alibaba.spring.boot</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
</dependency>

<!--zookeeper 注册中心-->
<dependency>
    <groupId>com.101tec</groupId>
    <artifactId>zkclient</artifactId>
</dependency>

<!--阿里解析 JSON 的 fastjson 依赖-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
</dependency>

<!--Apache commons-->
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
</dependency>

<!--Dom4j 依赖-->
<dependency>
    <groupId>org.dom4j</groupId>
    <artifactId>dom4j</artifactId>
</dependency>
```

```
<!--接口工程-->
<dependency>
    <groupId>com.abc.p2p</groupId>
    <artifactId>002-p2p-exterface</artifactId>
    <version>1.0.0</version>
</dependency>

<!--公共工程-->
<dependency>
    <groupId>com.abc.p2p</groupId>
    <artifactId>004-p2p-common</artifactId>
    <version>1.0.0</version>
</dependency>

<!--Google 生成二维码依赖-->
<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>core</artifactId>
</dependency>
<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>javase</artifactId>
</dependency>
```

3.7.22 添加日志文件

将日志文件 logback-spring.xml 添加到 resources 根目录下

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

```
<!-- 日志级别从低到高分为 TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为 WARN，则低于 WARN 的信息都不会输出 -->
<!-- scan:当此属性设置为 true 时，配置文件如果发生改变，将会被重新加载，默认值为 true -->
<!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当 scan 为 true 时，此属性生效。默认的时间间隔为 1 分钟。 -->
<!-- debug:当此属性设置为 true 时，将打印出 logback 内部日志信息，实时查看 logback 运行状态。默认值为 false。通常不打印 -->
<configuration  scan="true" scanPeriod="10 seconds">

    <!--输出到控制台-->
    <appender name="CONSOLE"
class="ch.qos.logback.core.ConsoleAppender">
        <!--此日志 appender 是为开发使用，只配置最低级别，控制台输出的日志级别是大于或等于此级别的日志信息-->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>debug</level>
        </filter>
        <encoder>
            <Pattern>%date [%-5p] [%thread] %logger{60}
[%file : %line] %msg%n</Pattern>
            <!-- 设置字符集 -->
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <appender name="FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
```

```
        <!--<File>/home/log/p2p-web.log</File>-->
        <File>D:/log/p2p-web.log</File>
        <encoder>
            <pattern>%date [%-5p] %thread %logger{60}
[%file : %line] %msg%n</pattern>
        </encoder>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 添加.gz 历史日志会启用压缩 大大缩小日志文件所占空间 -->

<!--<fileNamePattern>/home/log/p2p-web.log.%d{yyyy-MM-dd}.log</fileNa
mePattern>-->

<fileNamePattern>D:/log/p2p-web.log.%d{yyyy-MM-dd}.log</fileNamePatte
rn>
            <maxHistory>30</maxHistory><!--  保留 30 天日志 -->
        </rollingPolicy>
    </appender>



    <root level="INFO">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
    </root>
</configuration>
```

3.7.23 核心配置文件

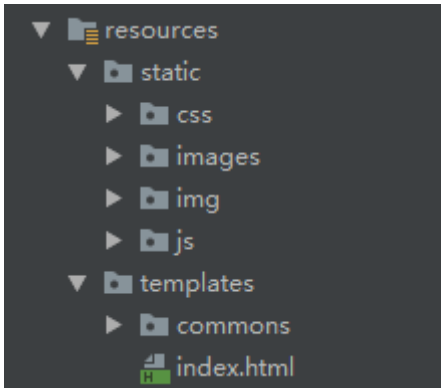
（10） application.properties

```
#设置服务端口号
server.port=8080
#设置上下文根
server.servlet.context-path=/p2p

#设置 thymeleaf 模版的缓存开关：关闭缓存
spring.thymeleaf.cache=false

#设置 thymeleaf 的前后缀
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html

#设置 Dubbo 服务消费者配置
spring.application.name=005-p2p-web
#设置注册中心
spring.dubbo.registry=zookeeper://localhost:2181

#设置项目请求响应字符编码
spring.http.encoding.enabled=true
spring.http.encoding.force=true
spring.http.encoding.charset=UTF-8
```

（11） application.yml

```yaml
server:
  port: 8080
  servlet:
    context-path: /p2p
spring:
  thymeleaf:
    cache: false
    prefix: classpath:/templates/
    suffix: .html
  http:
    encoding:
      enabled: true
      force: true
      charset: utf-8

  application:
    name: 005-p2p-web

  dubbo:
    registry: zookeeper://localhost:2181
```

3.7.24 导入前端静态资源文件



3.7.25 项目启动类

```
@SpringBootApplication

@EnableDubboConfiguration   //开启 Dubbo

public class Application {


    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);

    }

}
```

第4章 业务功能

4.1 P2P 首页展示

4.1.1 业务规则

1. 按照发布日期，新手宝理财产品只展示第 1 个
2. 按照发布日期，优先类理财产品展示前 4 个
3. 按照发布日期，散标类理财产品展示前 8 个
4. 显示平台历史年化收益率：所有理财产品年化收益的平均值
5. 平台注册总人数
6. 平台总投资金额
7. 历史年化收益率、注册总人数、总投资金额先从缓存中获取

4.2 分页展示产品信息

4.2.2 业务规则

1. 用户点击“查看更多 xxx 产品”跳转至该类产品分页显示页面，进行分页展示产品信息

4.3 展示商品详情页面

4.3.3 业务规则

1. 在首页面或分页展示产品页面中，用户点击“立即投资”跳转至产品详情页面
2. 在产品详情页面中，用户未登录时，不能进行投资操作
3. 根据用户投资时间，倒序显示该产品投资记录
4. 投资记录中，用户手机号保留前三位与最后两位

4.4 图片验证码

4.4.4 代码实现

（1） 代码实现

```
public static final int WIDTH = 120;
/**
 * 图片高度：50px
 */
public static final int HEIGHT = 50;
/**
 * 处理图片验证码方法
 *
 * @param request
 * @param response
 */
@RequestMapping(value="/jcaptcha/captcha")
public void handleCaptchaRequest(HttpServletRequest
request, HttpServletResponse response) {
```

```
        //生成 6 位随机验证码
        String captcha = this.getRandomCode(3);
    try {
        //创建字节数组输出流
        ByteArrayOutputStream jpegOutputStream = new
ByteArrayOutputStream();
        //创建图片缓存对象,BufferedImage.TYPE_INT_RGB :  表示一
个图像，该图像具有整数像素的 8 位 RGB 颜色
        BufferedImage bufferedImage = new
BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        //获取图片的画布
        Graphics graphics = bufferedImage.getGraphics();
        //设置画布背景色
        graphics.setColor(Color.GREEN);
        //设置画布填充区域
        graphics.fillRect(0, 0, WIDTH, HEIGHT);
        //边框区域
        graphics.drawRect(1, 1, WIDTH - 2, HEIGHT - 2);
        //设置字体颜色
        graphics.setColor(Color.BLACK);
        //设置字体样式

        graphics.setFont(new Font("微软雅黑", Font.ITALIC,
32));
        //填充数据
        graphics.drawString(captcha, 10, 38);
        //将生成的验证码存放到 session 中

request.getSession().setAttribute(Constants.CAPTCHA,
captcha);

        ImageIO.write(bufferedImage, "jpeg",
```

```
jpegOutputStream);
        byte[] captchaChallengeAsJpeg =
jpegOutputStream.toByteArray();

        //将验证码输出到页面
        response.setHeader("Cache-Control", "no-store");
        response.setHeader("Pragma", "no-cache");
        response.setDateHeader("Expires", 0L);
        response.setContentType("image/jpeg");
        ServletOutputStream respOs =
response.getOutputStream();
        respOs.write(captchaChallengeAsJpeg);
        respOs.flush();
        respOs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
/**
 * 生成 0-9, a-z,A-Z 的随机字符串
 * @param count
 * @return
 */
private String getRandomCode(int count) {
    String[] array =
{"A","B","C","D","E","F","G","H","I","J","K","L","M","N",
"O","P","Q","R","S","T","U","V","W","X","Y","Z","a","b","
c","d","e","f","g","h","i","j","k","l","m","n","o","p","q
","r","s","t","u","v","w","x","y","z","0","1","2","3","4"
,"5","6","7","8","9"};

    StringBuilder result = new StringBuilder();

    for (int i = 0; i < count; i++) {
        //生成一个下标 0-61 之间一个整数
        int index = (int) Math.round(Math.random() * 61);
        //从 array 数组中获取该下标字符放到 result 中
        result.append(array[index]);
    }
    return result.toString();
}
```

4.5注册功能

4.5.5 业务规则

1. 用户输入手机号必须进行验证
   a) 手机号不能为空
   b) 手机号格式
   c) 手机号是否已被注册
2. 密码验证格式：
   a) 密码不能为空
   b) 密码字符只可使用数字和大小写英文字母
   c) 密码应同时包含英文或数字
   d) 密码应为 6-16 位
   e) 两次输入密码必须一致
3. 再次验证登录密码
   a) 上次验证有错，隐藏提示信息
   b) 第一次密码是否为空
   c) 第二次密码是否为空
   d) 判断两次是否一致
4. 验证图片验证码：不能为空
5. 提交注册：需要所有验证都难过才可
6. 密码使用 JQuery 提供的 MD5 在页面进行加密
7. 注册必须使用 POST 请求
8. 验证仍然需要在后台服务端再次进行验证
9. 注册成功开立帐户

4.6不同系统交互方式：Http

参看 httpclient 项目代码

4.6.6 Java 原生 API

（2） 代码实现

```
/**
```

```
 * GET 请求
 * @param httpUrl
 * @return
 */
public static String doGet(String httpUrl) {
    HttpURLConnection connection = null;
    InputStream is = null;
    BufferedReader br = null;
    String result = null;//返回结果字符串

    try {
        //创建远程 url 连接对象
        URL url = new URL(httpUrl);
        //通过远程 url 连接对象打开一个连接，强转成 httpURLConnection
类
        connection = (HttpURLConnection) url.openConnection();
        //设置连接方式：get
        connection.setRequestMethod("GET");
        //设置连接主机服务器的超时时间：15000 毫秒
        connection.setConnectTimeout(15000);
        //设置读取远程返回的数据时间：60000 毫秒
        connection.setReadTimeout(60000);

        //通过 connection 连接，获取输入流
        is = connection.getInputStream();
        //封装输入流 is，并指定字符集
```

```
            br = new BufferedReader(new
InputStreamReader(is,"UTF-8"));

            //存放数据
            StringBuffer sbf = new StringBuffer();
            String temp = null;
            while ((temp = br.readLine()) != null) {
                sbf.append(temp);
                sbf.append("\r\n");//回车+换行
            }

            result = sbf.toString();

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭资源
            if(null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if(null != is) {
                try {
                    is.close();
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        connection.disconnect();//关闭远程连接
    }

    return result;
}


/**
 * POST 请求
 * @param httpUrl
 * @param param
 * @return
 */
public static String doPost(String httpUrl, String param) {
    HttpURLConnection connection = null;
    InputStream is = null;
    OutputStream os = null;
    BufferedReader br = null;
    String result = null;

    try {
        //创建远程 url 连接对象
        URL url = new URL(httpUrl);
```

```
        //通过远程 url 连接对象打开连接
        connection = (HttpURLConnection) url.openConnection();
        //设置连接请求方式
        connection.setRequestMethod("POST");
        //设置连接主机服务器超时时间：15000 毫秒
        connection.setConnectTimeout(15000);
        //设置读取主机服务器返回数据超时时间：60000 毫秒
        connection.setReadTimeout(60000);

        //默认值为：false，当向远程服务器传送数据/写数据时，需要设置为
true
        connection.setDoOutput(true);
        //默认值为：true，当前向远程服务读取数据时，设置为 true，该参
数可有可无
        connection.setDoInput(true);

        //通过连接对象获取一个输出流
        os = connection.getOutputStream();
        //通过输出流对象将参数写出去/传输出去,它是通过字节数组写出的
        os.write(param.getBytes());//把需要传送的参数发送给远程
url

        //通过连接对象获取一个输入流，向远程读取
        is = connection.getInputStream();
        //对输入流对象进行包装
```

```
            br = new BufferedReader(new InputStreamReader(is,
"UTF-8"));

            StringBuffer sbf = new StringBuffer();
            String temp = null;
            //循环遍历一行一行读取数据
            while ((temp = br.readLine()) != null) {
                sbf.append(temp);
                sbf.append("\r\n");
            }

            result = sbf.toString();



        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            //关闭资源
            if(null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if(null != os) {
```

```
        try {
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    if(null != is) {
        try {
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //断开与远程地址 url 的连接
    connection.disconnect();
  }
  return result;
}
```

4.6.7 HttpClient 3.1

（3） 添加依赖

```
<!-- httpclient3.1 版本 -->
<dependency>
    <groupId>commons-httpclient</groupId>
```

```
    <artifactId>commons-httpclient</artifactId>
    <version>3.1</version>
</dependency>
```

（4） 代码实现

```
/**
 * GET 请求方法
 * 注：如果需要传递参数，把参数拼接在 url 地址后面
 * @param url
 */
public static String doGet(String url) {
    //输入流
    InputStream is = null;
    BufferedReader br = null;
    String result = null;

    //创建 httpClient 实例
    HttpClient httpClient = new HttpClient();

    //设置 http 连接主机服务超时时间：15000 毫秒
    //先获取连接管理器对象，再获取参数对象,再进行参数的赋值

httpClient.getHttpConnectionManager().getParams().setConn
ectionTimeout(15000);
```

```
    //创建一个 Get 方法实例对象
    GetMethod getMethod = new GetMethod(url);
    //设置 get 请求超时为 60000 毫秒

getMethod.getParams().setParameter(HttpMethodParams.SO_TI
MEOUT, 60000);
    //设置请求重试机制，默认重试次数：3 次，参数设置为 true，重试机制
可用，false 相反

getMethod.getParams().setParameter(HttpMethodParams.RETRY
_HANDLER, new DefaultHttpMethodRetryHandler(1, true));


    try {
        //执行 Get 方法
        int statusCode = httpClient.executeMethod(getMethod);
        //判断返回码
        if(statusCode != HttpStatus.SC_OK) {
            //如果状态码返回的不是 ok,说明失败了,打印错误信息
            System.err.println("Method faild: " +
getMethod.getStatusLine());
        }

        //通过 getMethod 实例，获取远程的一个输入流
        is = getMethod.getResponseBodyAsStream();
        //包装输入流
```

```
            br = new BufferedReader(new InputStreamReader(is,
"UTF-8"));

            StringBuffer sbf = new StringBuffer();

            //读取封装的输入流

            String temp = null;

            while ((temp = br.readLine()) != null) {

                sbf.append(temp).append("\r\n");

            }

            result = sbf.toString();

        } catch (Exception e) {

            System.err.println("Fatal protocol violation: " +
e.getMessage());

            e.printStackTrace();

        } finally {

            //关闭资源

            if(null != br) {

                try {

                    br.close();

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

            if(null != is) {

                try {
```

```
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //释放连接
    getMethod.releaseConnection();
  }

  return result;
}

/**
 * POST 请求方法
 * @param url
 * @param paramMap
 * @return
 * @throws UnsupportedEncodingException
 */
public static String doPost(String url, Map<String,Object>
paramMap) throws UnsupportedEncodingException {
  //获取输入流
  InputStream is = null;
  BufferedReader br = null;
  String result = null;

  //创建 httpClient 实例对象
```

```
    HttpClient httpClient = new HttpClient();
    //设置 httpClient 连接主机服务器超时时间：15000 毫秒

httpClient.getHttpConnectionManager().getParams().setConn
ectionTimeout(15000);

    //创建 post 请求方法实例对象
    PostMethod postMethod = new PostMethod(url);
    //设置 post 请求超时时间

postMethod.getParams().setParameter(HttpMethodParams.SO_T
IMEOUT, 60000);

    NameValuePair[] nvp = null;
    //判断参数 map 集合 paramMap 是否为空
    if(null != paramMap && paramMap.size() > 0) {//不为空
        //创建键值参数对象数组，大小为参数的个数
        nvp = new NameValuePair[paramMap.size()];
        //循环遍历参数集合 map
        Set<Entry<String, Object>> entrySet =
paramMap.entrySet();
        //获取迭代器
        Iterator<Entry<String, Object>> iterator =
entrySet.iterator();

        int index = 0;
```

```
        while(iterator.hasNext()) {
            Entry<String, Object> mapEntry = iterator.next();
            //从 mapEntry 中获取 key 和 value 创建键值对象存放到数组中
            nvp[index] = new NameValuePair(mapEntry.getKey(),
new
String(mapEntry.getValue().toString().getBytes("UTF-8"),"
UTF-8"));
            index++;
        }
    }

    //判断 nvp 数组是否为空
    if(null != nvp && nvp.length > 0) {
        //将参数存放到 requestBody 对象中
        postMethod.setRequestBody(nvp);
    }

    try {
        //执行 POST 方法
        int statusCode = httpClient.executeMethod(postMethod);
        //判断是否成功
        if(statusCode != HttpStatus.SC_OK) {
            System.err.println("Method faild: " +
postMethod.getStatusLine());
        }

        //获取远程返回的数据
```

```
            is = postMethod.getResponseBodyAsStream();
            //封装输入流
            br = new BufferedReader(new InputStreamReader(is,
"UTF-8"));

            StringBuffer sbf = new StringBuffer();
            String temp = null;
            while ((temp = br.readLine()) != null) {
                sbf.append(temp).append("\r\n");
            }

            result = sbf.toString();

        } catch (Exception e) {
            System.err.println("Fatal protocol violation: " +
e.getMessage());
            e.printStackTrace();
        } finally {

            //关闭资源
            if(null != br) {
                try {
                    br.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

```

```
        if(null != is) {
            try {
                is.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        //释放连接
        postMethod.releaseConnection();
    }
    return result;
}
```

4.6.8 HttpClient 4.5

（5） 添加依赖

```
<!-- httpclient4.5版本 -->
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.3</version>
</dependency>
```

（6） 代码实现

```
/**
```

```
 * Get 请求方法
 *
 * @param url
 * @return
 */
public static String doGet(String url) {
    CloseableHttpClient httpClient = null;
    CloseableHttpResponse response = null;
    String result = "";

    try {
        //通过址默认配置创建一个 httpClient 实例
        httpClient = HttpClients.createDefault();
        //创建 httpGet 远程连接实例
        HttpGet httpGet = new HttpGet(url);
        //设置配置请求参数
        RequestConfig requestConfig = RequestConfig.custom()
                .setConnectTimeout(35000)//连接主机服务超时时间
                .setConnectionRequestTimeout(35000)//请求超时时间
                .setSocketTimeout(60000)//数据读取超时时间
                .build();
        //为 httpGet 实例设置配置
        httpGet.setConfig(requestConfig);
        //执行 get 请求得到返回对象
        response = httpClient.execute(httpGet);
        //通过返回对象获取返回数据
        HttpEntity entity = response.getEntity();
        //通过 EntityUtils 中的 toString 方法将结果转换为字符串
        result = EntityUtils.toString(entity);

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
```

```
        //关闭资源
        if (null != response) {
            try {
                response.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        if (null != httpClient) {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    return result;
}

public static String doPost(String url, Map<String, Object>
paramMap) {
    CloseableHttpClient httpClient = null;
    CloseableHttpResponse response = null;
    String result = "";

    try {
        //创建 httpClient 实例
        httpClient = HttpClients.createDefault();
        //创建 httpPost 远程连接实例
        HttpPost httpPost = new HttpPost(url);
        //配置请求参数实例
        RequestConfig requestConfig = RequestConfig.custom()
                .setConnectTimeout(35000)//设置连接主机服务超时时间
                .setConnectionRequestTimeout(35000)//设置连接请求
超时时间
```

```
            .setSocketTimeout(60000)//设置读取数据连接超时时间
            .build();
        //为 httpPost 实例设置配置
        httpPost.setConfig(requestConfig);

        //封装 post 请求参数
        if (null != paramMap && paramMap.size() > 0) {
            List<NameValuePair> nvps = new
ArrayList<NameValuePair>();
            //通过 map 集成 entrySet 方法获取 entity
            Set<Entry<String, Object>> entrySet =
paramMap.entrySet();
            //循环遍历，获取迭代器
            Iterator<Entry<String, Object>> iterator =
entrySet.iterator();
            while (iterator.hasNext()) {
                Entry<String, Object> mapEntry = iterator.next();
                nvps.add(new
BasicNameValuePair(mapEntry.getKey(),
mapEntry.getValue().toString()));
            }

            //为 httpPost 设置封装好的请求参数
            httpPost.setEntity(new UrlEncodedFormEntity(nvps,
"UTF-8"));
        }

        //执行 post 请求得到返回对象
        response = httpClient.execute(httpPost);
        //通过返回对象获取数据
        HttpEntity entity = response.getEntity();
        //将返回的数据转换为字符串
        result = EntityUtils.toString(entity);
    } catch (Exception e) {
        e.printStackTrace();
```

```
    } finally {
        //关闭资源
        if (null != response) {
            try {
                response.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        if (null != httpClient) {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return result;
}
```

4.7 实名认证

4.7.9 业务规则

1. 验验证真实姓名格式
   a) 姓名不能为空
   b) 真实姓名只能输入中文
2. 验证身份证号码：只能通过后台服务进行验证
   a) 身份号码不能为空
   b) 验证身份证号码格式
   c) 身份证号码应为 15 或 18 位
   d) 两次输入的身份证号是否一致
3. 验证再次输入身份号码
   a) 如果上次验证没过，隐藏错误信息
   b) 第一次输入不能为空
   c) 第二次输入不能为空
   d) 判断两次是否一致
4. 验证图形验证码

5. 服务端需要再次进行验证
6. 调用第三方系统接口，认证用户身份
7. 认证成功跳转至个人中心页面

4.8平台统计信息

4.8.10 业务规则

1. 平台总用户数
2. 平台总投资金额
3. 平台历史年化收益率

4.9用户登录

4.9.11 业务规则

1. 验证手机号格式
   a) 手机号不能为空
   b) 手机号格式
2. 登录密码不能为空
3. 验证图片验证码
4. 后台服务端需要再次验证
5. 验证全部通过才可提交登录

4.10 个人中心

4.10.12 业务规则

1. 展示用户基本信息
   a) 如果用户未实名认证，点击“实名认证”
2. 展示用户财务信息
3. 按投资时间倒序，展示最近 5 笔投资记录，可点击“查看全部投资记录”
4. 按充值时间倒序，展示最近 5 笔充值记录，可点击“查看全部充值记录”
5. 按收益时间倒序，展示最近 5 笔收益记录，可点击“查看全部收益记录”

4.11 个人中心-分页查询投资记录

4.11.13 业务规则

1. 分页查询投资记录，按照投资时间倒序展示

4.12 个人中心-分页查询充值记录

4.12.14 业务规则

1. 分页查询充值记录：按照充值时间倒序展示

4.13 个人中心-分页查询收益记录

4.13.15 业务规则

1. 分页查询收益记录：按照收益时间倒序展示

4.14 用户退出

4.14.16 业务规则

1. 将 session 中的用户信息和账户信息清空

4.15 我的账户

4.15.17 业务规则

1. 用户未认证，姓名和身份号码显示“请认证”，已认证，显示用户真实姓名和身份证号码

4.16 投资

4.16.18 业务规则

1. 验证用户输入投资金额
   a) 投资金额不能为空
   b) 投资金额应为大于 0 的整数
   c) 投资金额应为 100 的整数
   d) 投资金额不能低于起投金额
   e) 单笔投资金额不能超过最大投资限额
   f) 验证通过计算预计本息收益
2. 计算预计本息收益，公式如下：

   本息收益 = 投资金额 * （年化利率/100/365）* 投资周期

   投资周期：单位为天
3. 投资请求
   a) 产品是否已经满标
   b) 验证用户输入投资金额，即 1
   c) 验证用户是否登录，未登录，点击“立即投资”跳转至用户登录页面
   d) 验证用户是否实名认证，未认证，点击“认证”跳转至实名认证页面
   e) 判断投资金额是否大于账户余额
4. 后台判断用户账户余额是否充足
5. 用户未登录，资金显示：请登录
6. 投资业务
   a) 更新产品剩余可投金额
   b) 更新账户金额
   c) 新增投资记录
   d) 更新投资排行榜
   e) 更新平台总投资金额
7. 更新 session 中账户信息

4.17 查询投资排行榜

4.18 生成收益计划

4.19 生成收益回款

4.20 充值功能

4.21 拦截未登录请求