

03 Java基础-框架整合

- 03 Java基础-框架整合
 - 项目构建
 - 项目结构
 - 创建父工程
 - 创建子模块
 - 整理父 pom 文件中的内容
 - 常见错误
 - Maven Profile 多环境打包
 - Mybatis Plus
 - springboot 集成mybatis plus
 - mybatis plus 代码生成器
 - mybatis plus active record
 - mybatis plus 优化
 - 数据库连接池
 - HikariCP
 - 动态数据源实战

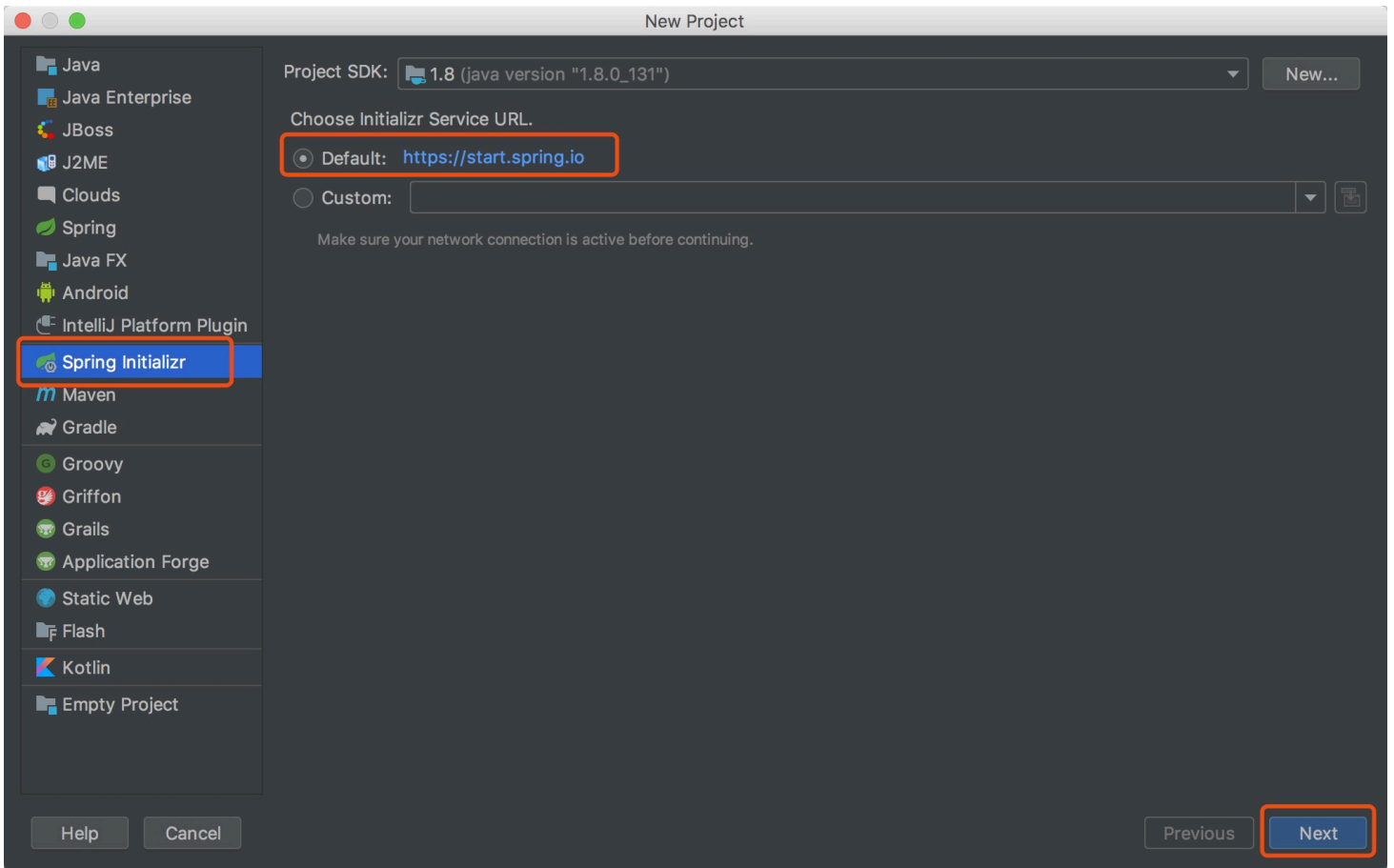
项目构建

项目结构

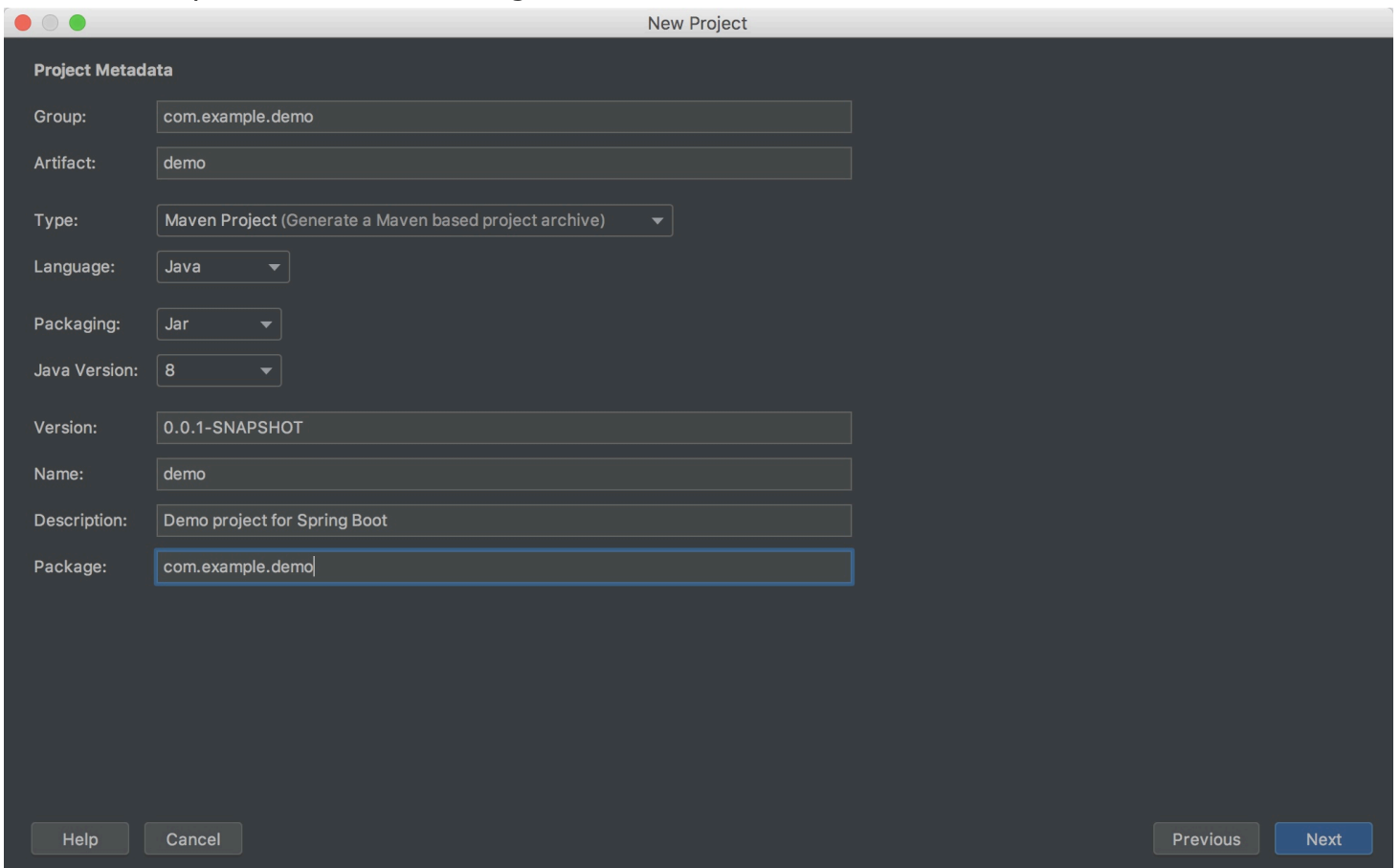
1. service 层（业务逻辑层）
2. dao 层（数据持久层）
3. common 层（公用组件层）
4. web 层（请求处理层）
5. 注：service 层依赖 dao 及 common 层， web 层依赖 service 层

创建父工程

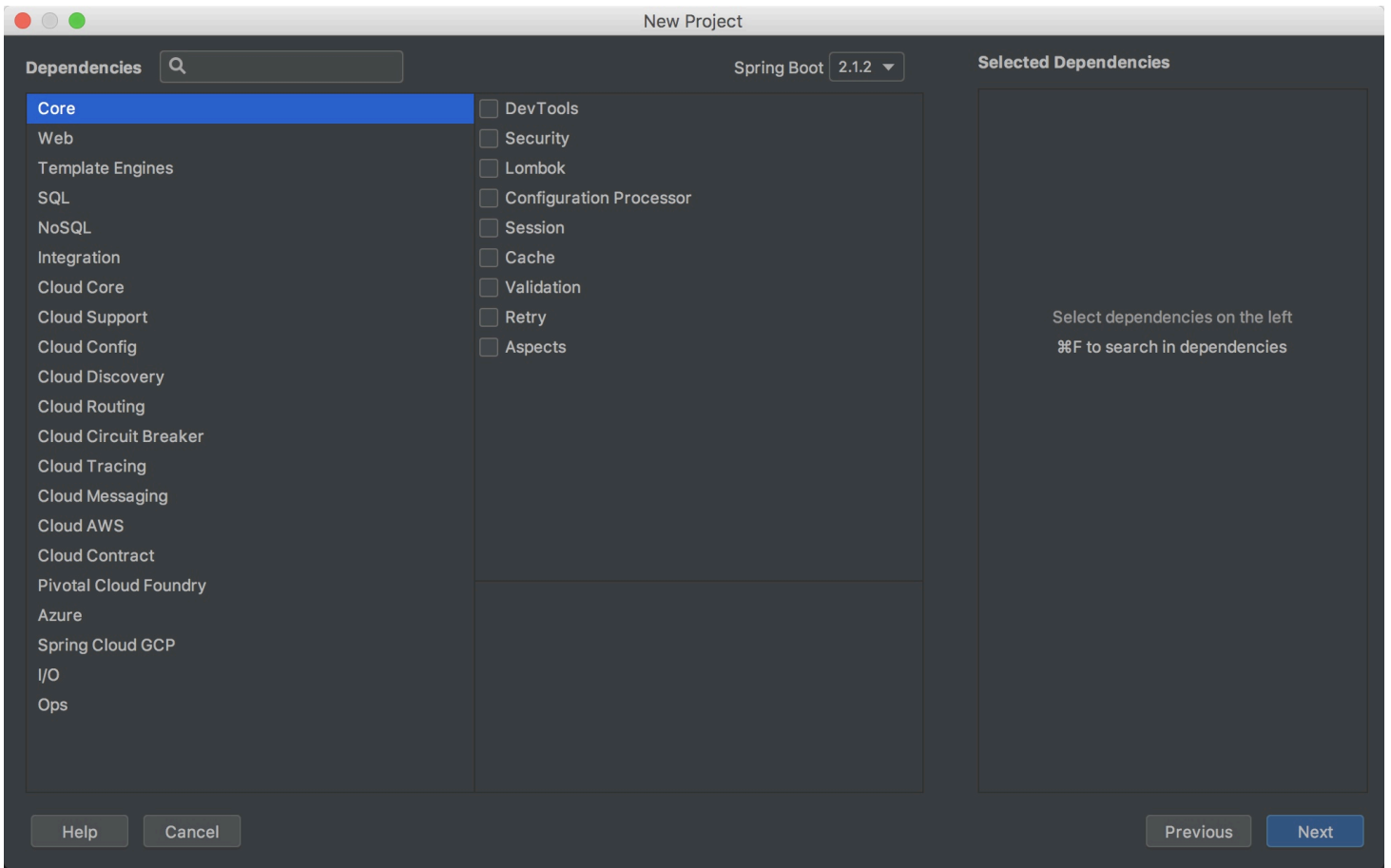
- ① IDEA 主面板选择菜单「Create New Project」或者工具栏选择菜单「File -> New -> Project...」
- ② 侧边栏选择「Spring Initializr」， Initializr 默认选择 Default，然后点击「Next」



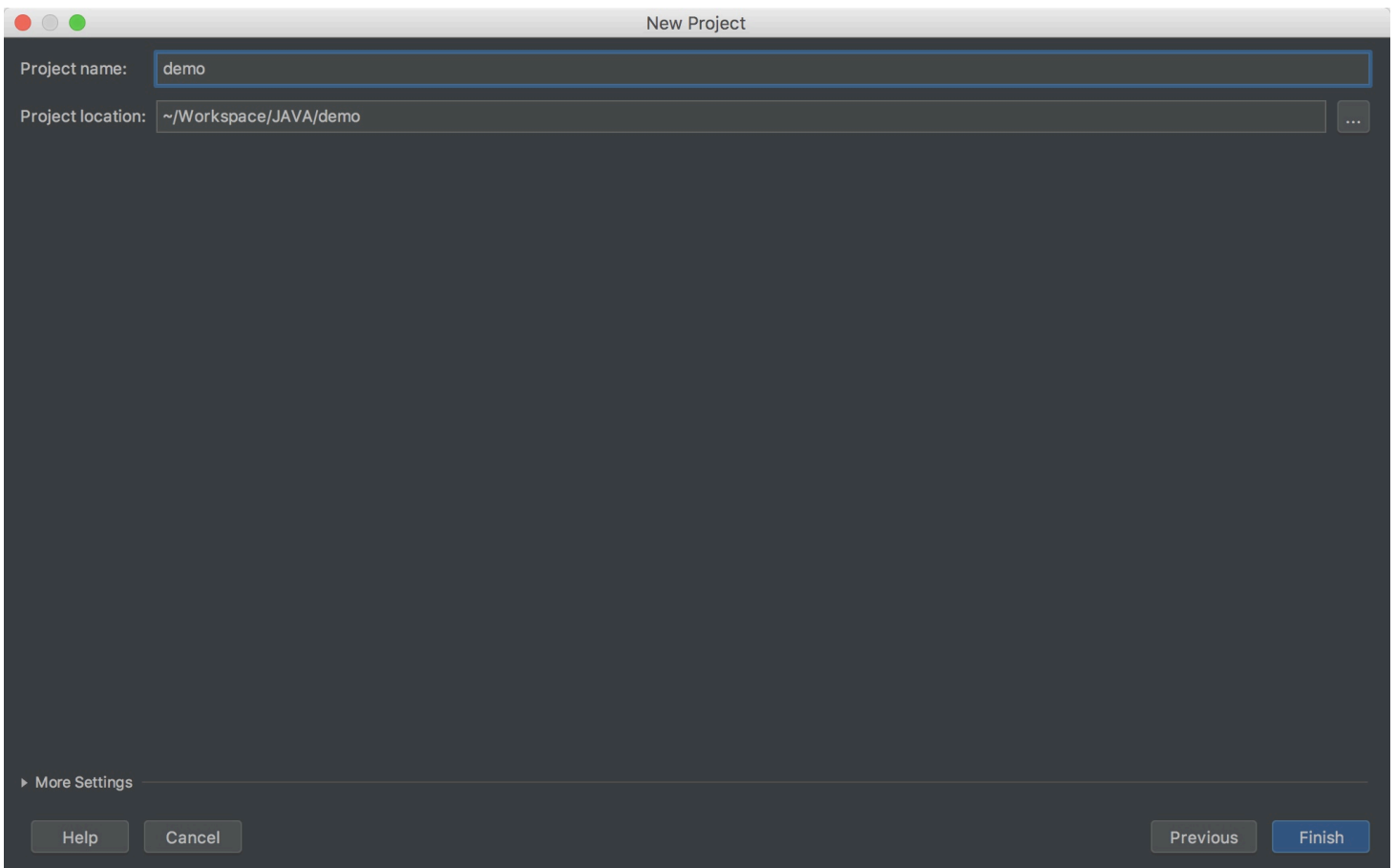
③ 修改 Group 、 Artifact 、 Package 输入框中的值后点击 「 Next 」



④ 这步暂时先不需要选择，直接点 「 Next 」



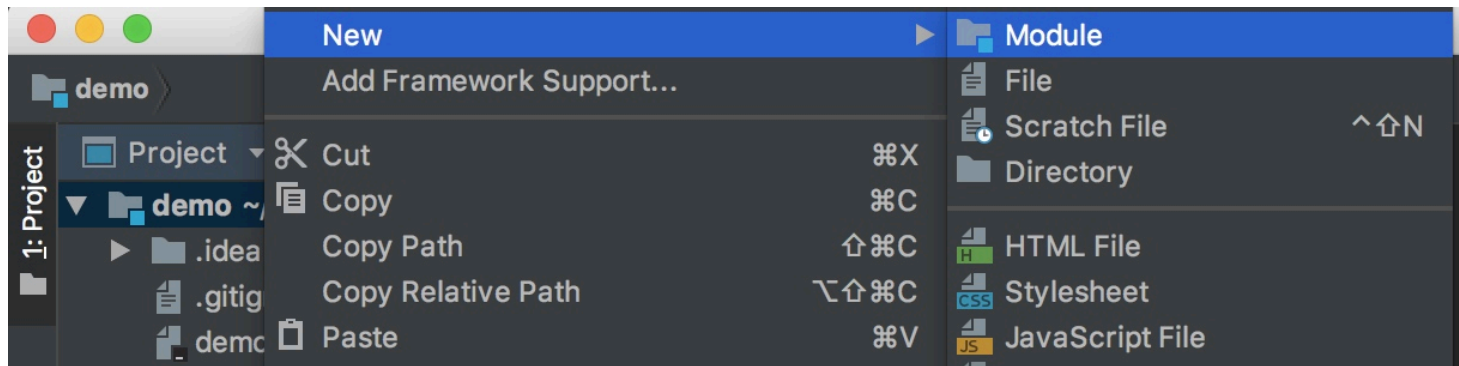
⑤ 点击「Finish」创建项目



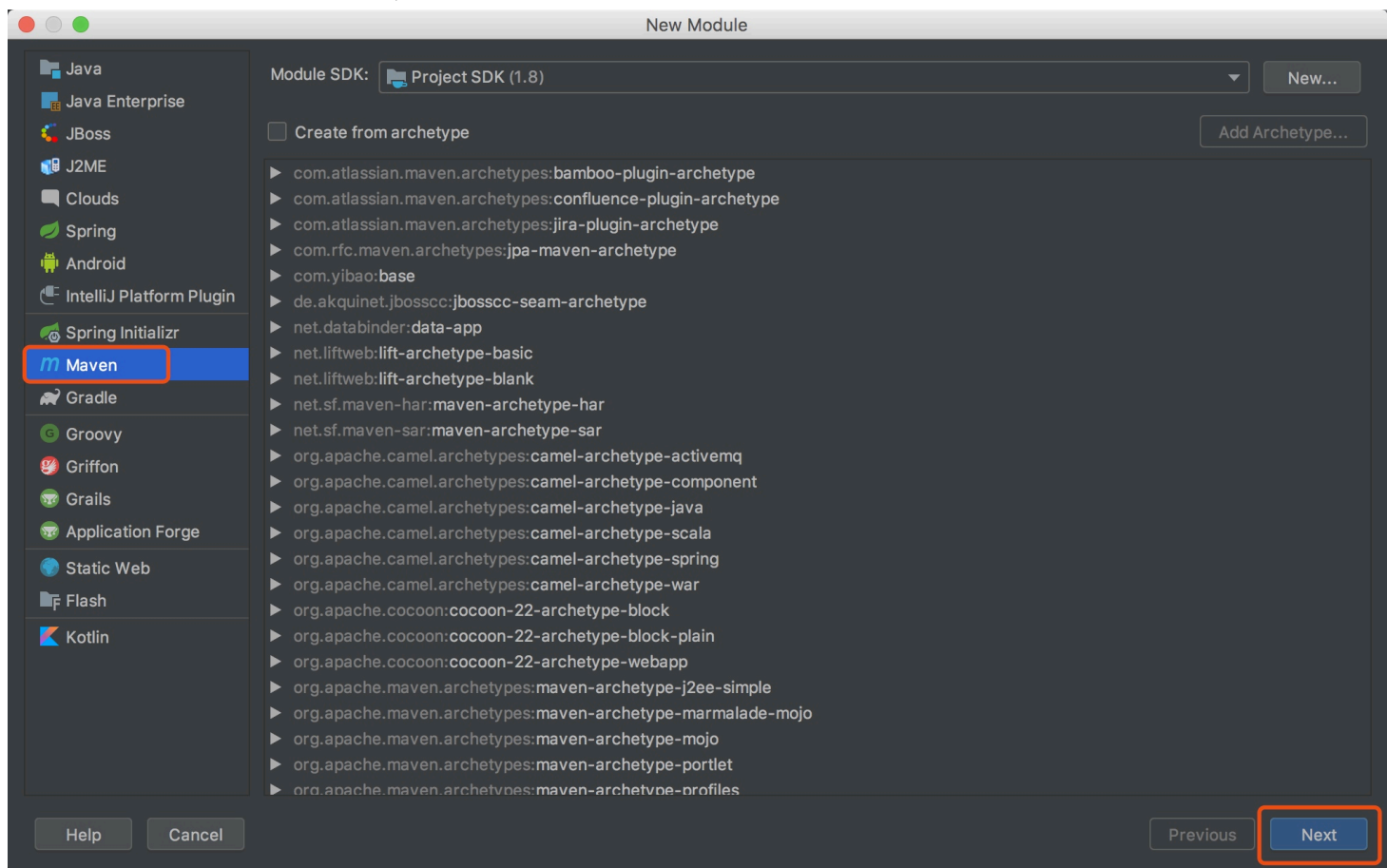
⑥ 删除无用的 .mvn 目录、src 目录、mvnw 及 mvnw.cmd 文件，最终只留 .gitignore 和 pom.xml

创建子模块

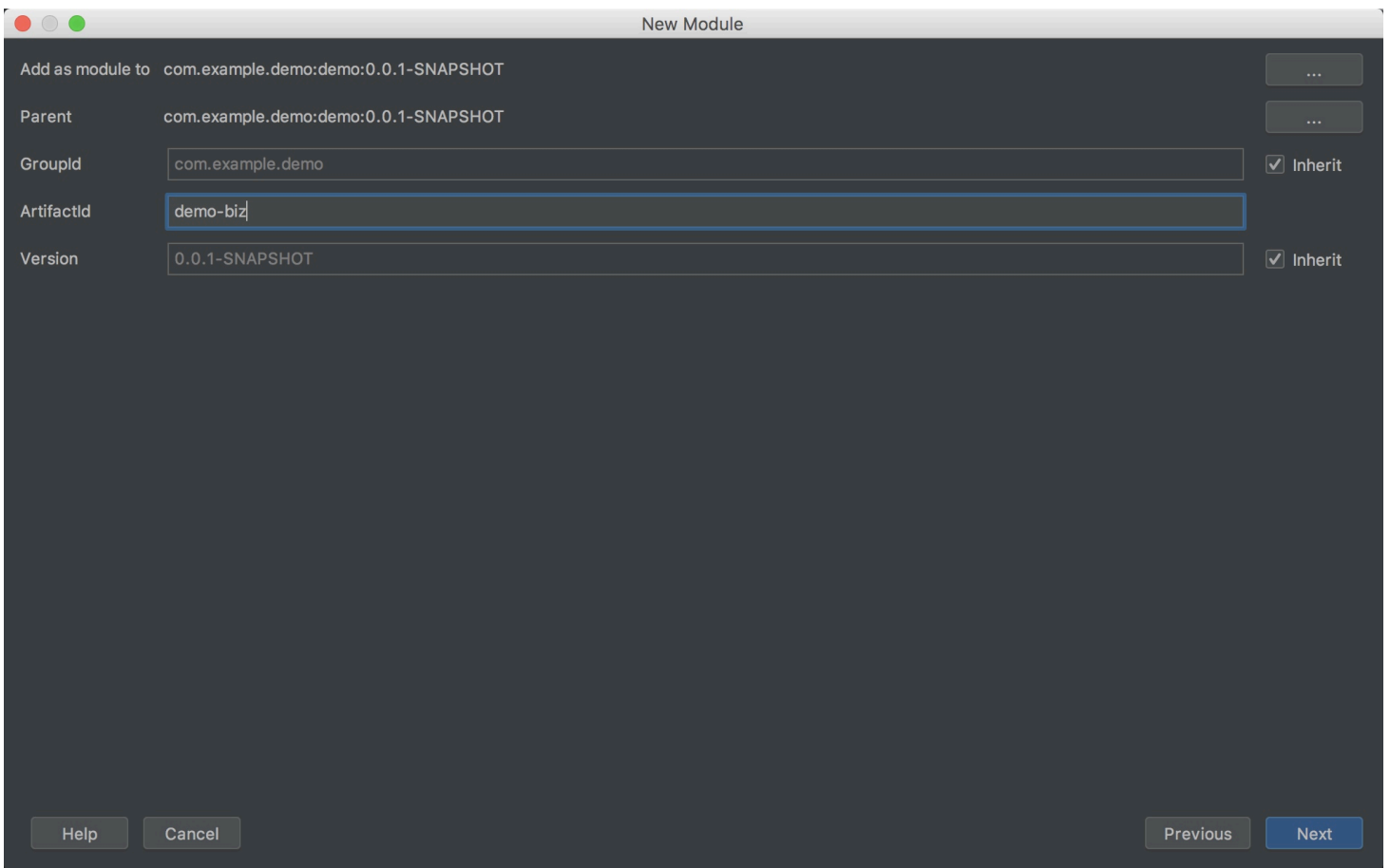
① 选择项目根目录，右键呼出菜单，选择「New -> Module」



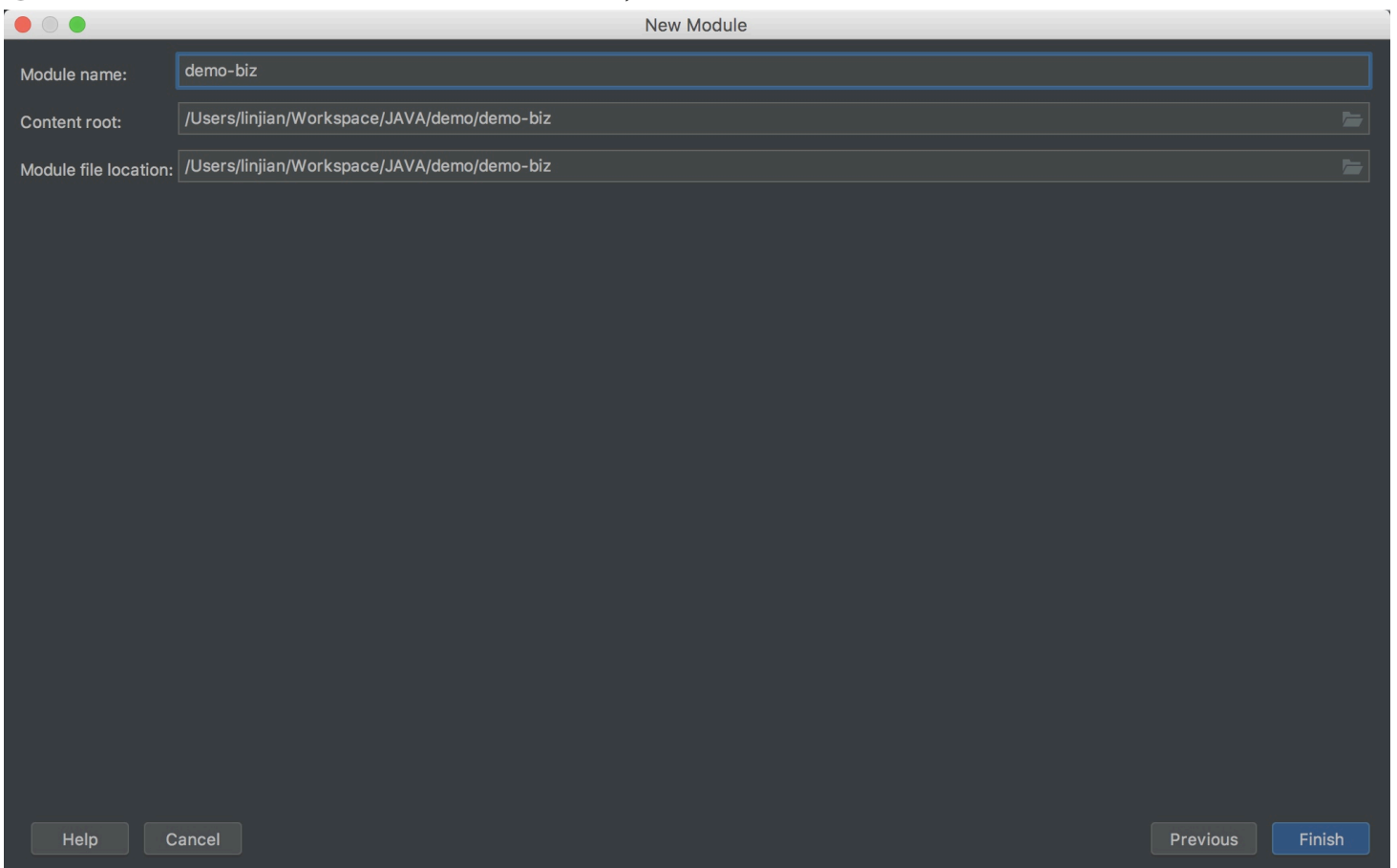
② 侧边栏选择「Maven」，点击「Next」



③ 填写 ArifactId，点击「Next」



④ 修改 Module name 增加横杠提升可读性，点击「Finish」



⑤ 同理添加「demo-dao」、「demo-common」、「demo-web」子模块

整理父 pom 文件中的内容

① 删除 dependencies 标签及其中的 spring-boot-starter 和 spring-boot-starter-test 依赖，因为 Spring Boot 提供的父工程已包含，并且父 pom 原则上都是通过 dependencyManagement 标签管理依赖包。

② 删除 build 标签及其中的所有内容，spring-boot-maven-plugin 插件作用是打一个可运行的包，多模块项目仅仅需要在入口类所在的模块添加打包插件，这里父模块不需要打包运行。而且该插件已被包含在 Spring Boot 提供的父工程中，这里删掉即可。

③ dependencies与dependencyManagement

1. dependencyManagement Maven 使用dependencyManagement 元素来提供了一种管理依赖版本号的方式。通常会在一个组织或者项目的最顶层的父POM 中看到 dependencyManagement 元素;这样做的好处：统一管理项目的版本号，确保应用的各个项目的依赖和版本一致，才能保证测试的和发布的是相同的成果，因此，在顶层pom中定义共同的依赖关系。同时可以避免在每个使用的子项目中都声明一个版本号，这样想升级或者切换到另一个版本时，只需要在父类容器里更新，不需要任何一个子项目的修改；如果某个子项目需要另外一个版本号时，只需要在dependencies中声明一个版本号即可。子类就会使用子类声明的版本号，不继承于父类版本号。
2. dependencies 相对于dependencyManagement，所有生命在dependencies里的依赖都会自动引入，并默认被所有的子项目继承。

常见错误

Consider defining a bean of type '***' in your configuration

设置 @SpringBootApplication 注解中的 scanBasePackages 值为

Maven Profile 多环境打包

在日常开发中，通常不止一套环境，如开发环境、测试环境、预发环境、生成环境，而每个环境的配置项可能都不一样，这就需要用到多环境打包来解决这个问题。

① 在 demo-web 层的 resources 目录中新建 conf 目录，再在其中按照环境创建相应目录，这里创建开发环境「dev」及测试环境「test」，再将原本的 application.properties 文件分别拷贝一份到两个目录中，根据环境修改其中的配置项，最后删除原本的配置文件。得到目录结构如下：

```
|-- resources
    |-- conf
    |   |-- dev
    |       |-- application.properties
    |-- test
        |-- application.properties
```

② 往 demo-web 层的 pom 文件添加 profile 标签

```
<profiles>
  <profile>
    <id>dev</id>
    <properties>
      <profile.env>dev</profile.env>
    </properties>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
  </profile>
  <profile>
    <id>test</id>
    <properties>
      <profile.env>test</profile.env>
    </properties>
  </profile>
</profiles>
```

注：其中 dev 为默认激活的 profile，如要增加其他环境按照上述步骤操作即可。

③ 设置打包时资源文件路径

```
<build>
  <finalName>demo</finalName>
  <resources>
    <resource>
      <directory>${basedir}/src/main/resources</directory>
      <excludes>
        <exclude>conf/**</exclude>
      </excludes>
    </resource>
    <resource>
      <directory>src/main/resources/conf/${profile.env}</directory>
```

```
        </resource>
    </resources>
</build>
```

注：\${basedir} 为当前子模块的根目录

④ 打包时通过「P」参数指定 profile

mvn clean install -P test

Mybatis Plus

MyBatis 是一个java的半自动化ORM框架，而MyBatis-Plus是对它的二次封装，简化MyBatis的使用，简单的说，可以尽可能少的手写增删改查的sql语句。

springboot 集成mybatis plus

- 添加jar包
- 添加扫描包配置类
- 修改配置文件

mybatis plus 代码生成器

MyBatis-Plus 提供了代码生成器的功能，能自动帮我们生成mapper.xml、Mapper.java POJO、dao、service、controller 等文件。

mybatis plus active record

MP提供了ActiveRecord的支持，所以实体类只需继承 Model 类即可实现基本 CRUD 操作。

mybatis plus 优化

分页插件

日志: logback

参数: properties

数据库连接池

1. 使用数据库连接池的原因

- i. 所有数据库连接池都遵守基本的设计规则，实现 `javax.sql.DataSource` 接口，里面最重要的方法就是 `Connection getConnection() throws SQLException`；用于获取一个 `Connection`，一个 `Connection` 就是一个数据库链接，就是一个 TCP 链接，建立 TCP 链接是需要进行 3 次握手的，这降低来链接的使用效率，也是各种数据库连接池存在的原因
- ii. 数据库连接池通过事先建立好 `Connection` 并缓存起来，这样应用需要做数据查询的时候，直接从缓存中拿到 `Connection` 就可以使用来。数据库连接池还能够检测异常的链接，释放闲置的链接。

HikariCP

1. 特性

spring boot2.0 已经将 HikariCP 做为了默认的数据源连接池。

官网详细地说明了 HikariCP 所做的一些优化，总结如下：

- i. 字节码精简：优化代码，直到编译后的字节码最少，这样，CPU 缓存可以加载更多的程序代码；
- ii. 优化代理和拦截器：减少代码，例如 HikariCP 的 `Statement proxy` 只有 100 行代码，只有 BoneCP 的十分之一；
- iii. 自定义数组类型（`FastStatementList`）代替 `ArrayList`：避免每次 `get()` 调用都要进行 `range check`，避免调用 `remove()` 时的从头到尾的扫描；
- iv. 自定义集合类型（`ConcurrentBag`）：提高并发读写的效率；
- v. 稳定性

通过测试 `getConnection()`，各种 CP 的不相同处理方法如下：

- HikariCP：等待 5 秒钟后，如果连接还是没有恢复，则抛出一个 `SQLExceptions` 异常；后续的 `getConnection()` 也是一样处理；
- C3P0：完全没有反应，没有提示，也不会在“`CheckoutTimeout`”配置的时长超时后有任何通知给调用者；然后等待 2 分钟后终于醒来了，返回一个 `error`；
- Tomcat：返回一个 `connection`，然后.....调用者如果利用这个无效的 `connection` 执行 SQL 语句.....结果可想而知；大约 55 秒之后终于醒来了，这时候的 `getConnection()` 终于可以返回一个 `error`，但没有等待参数配置的 5 秒钟，而是立即返回 `error`；
- BoneCP：跟 Tomcat 的处理方法一样；也是大约 55 秒之后才醒来，有了正常的反应，并且终于会等待 5 秒钟之后返回 `error` 了；

1. 配置

HikariCP连接池连接数

连接数 = ((核心数 * 2) + 有效磁盘数)

小型4核i7服务器连接池大小设置为： $9 = ((4 * 2) + 1)$ 。这样的连接池大小居然可以轻松处理3000个前端用户在6000 TPS下运行简单查询。

动态数据源实战

源码链接：<https://gitlab.com/Genesisx/zhuawacourse/-/tree/master/frame/design/src/main/java>

这个代码是整个课程的，不是每节课一个代码，整个框架是所有代码的基础，不是单单一个文件夹