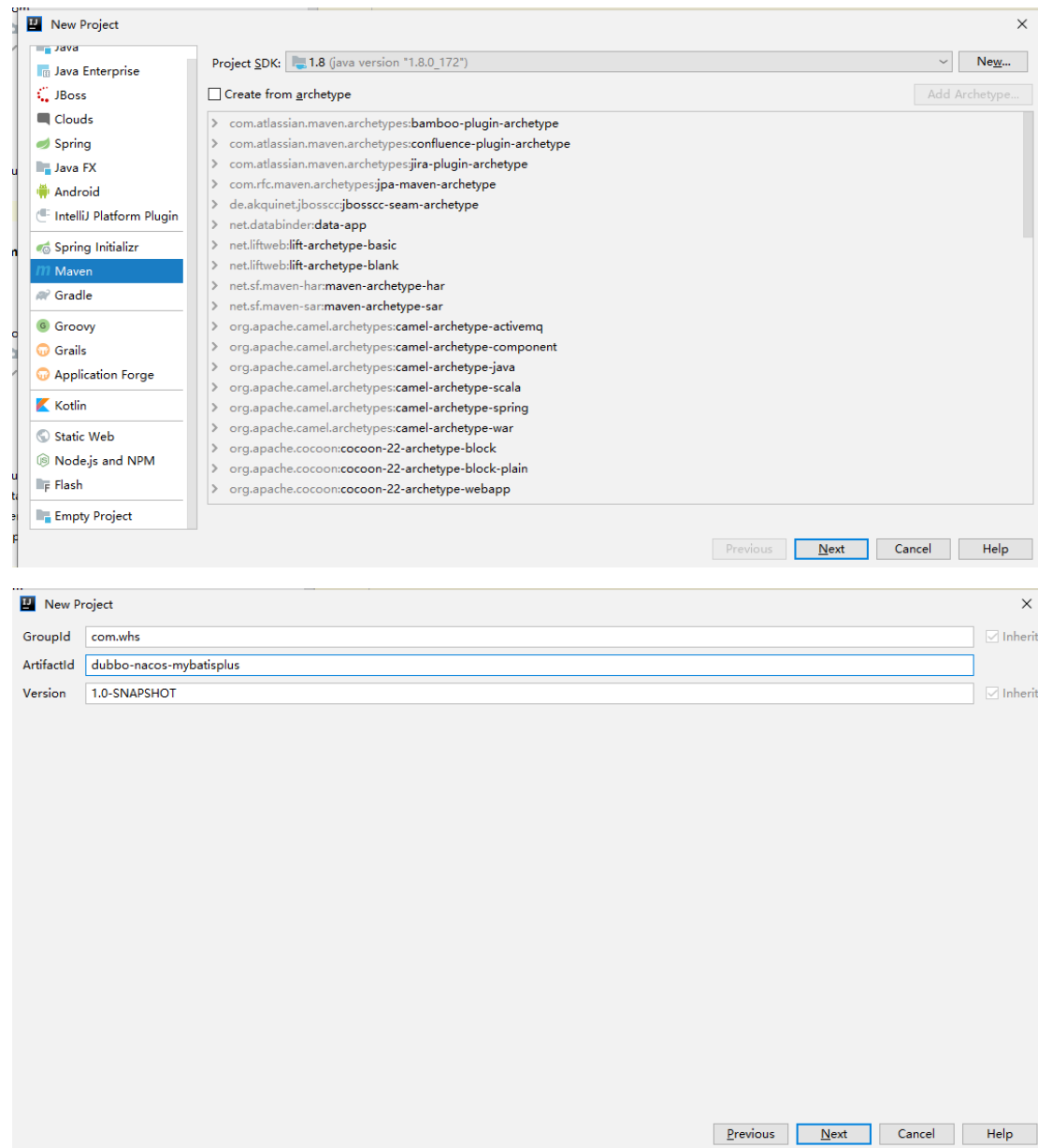


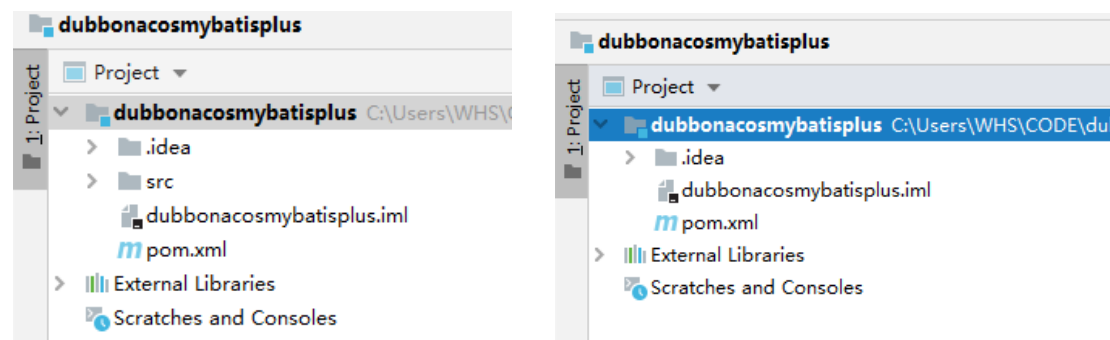
目录

目录	1
一、创建 maven 工程.....	2
二、创建 dubbo-nacos-mybatisplus-provider 子模块	3
2.1 创建	3
2.2 添加依赖.....	5
2.3 application.yml 配置	5
2.4 mybatisplus 逆向工程	6
2.5 修改代码.....	7
三、创建 dubbo-nacos-mybatisplus-api 子模块	8
3.1 创建	8
3.2 添加 java 代码	9
四、创建 dubbo-nacos-mybatisplus-consumer 子模块	11
4.1 创建	11
4.2 添加依赖.....	13
4.3 application.yml 配置	13
4.4 添加 java 代码	14
五、模块之间的依赖配置.....	15
5.1 模块打包.....	15
5.2 添加依赖.....	16
六、添加代码.....	17
6.1 dubbo-nacos-mybatisplusr-api 接口声明函数.....	17
6.2 dubbo-nacos-mybatisplusr-provider 函数实现	17
6.2.1 PersonDao 函数声明以及 PersonDao.xml 的 sql	17
6.2.2 PersonServiceImpl 接口函数实现	18
6.3 dubbo-nacos-mybatisplusr-consumer 调用	18
项目结构.....	19
项目运行.....	20
1. 启动 nacos	20
2. 运行启动类.....	20
3. 测试.....	20
附件	22
附件 1 dubbo-nacos-mybatisplusr-provider 子模块的依赖	22
附件 2 CodeGeneration 代码	23

一、创建 maven 工程



项目创建成功后，项目结构如下左图。删除 src 包，项目结构如下右图。



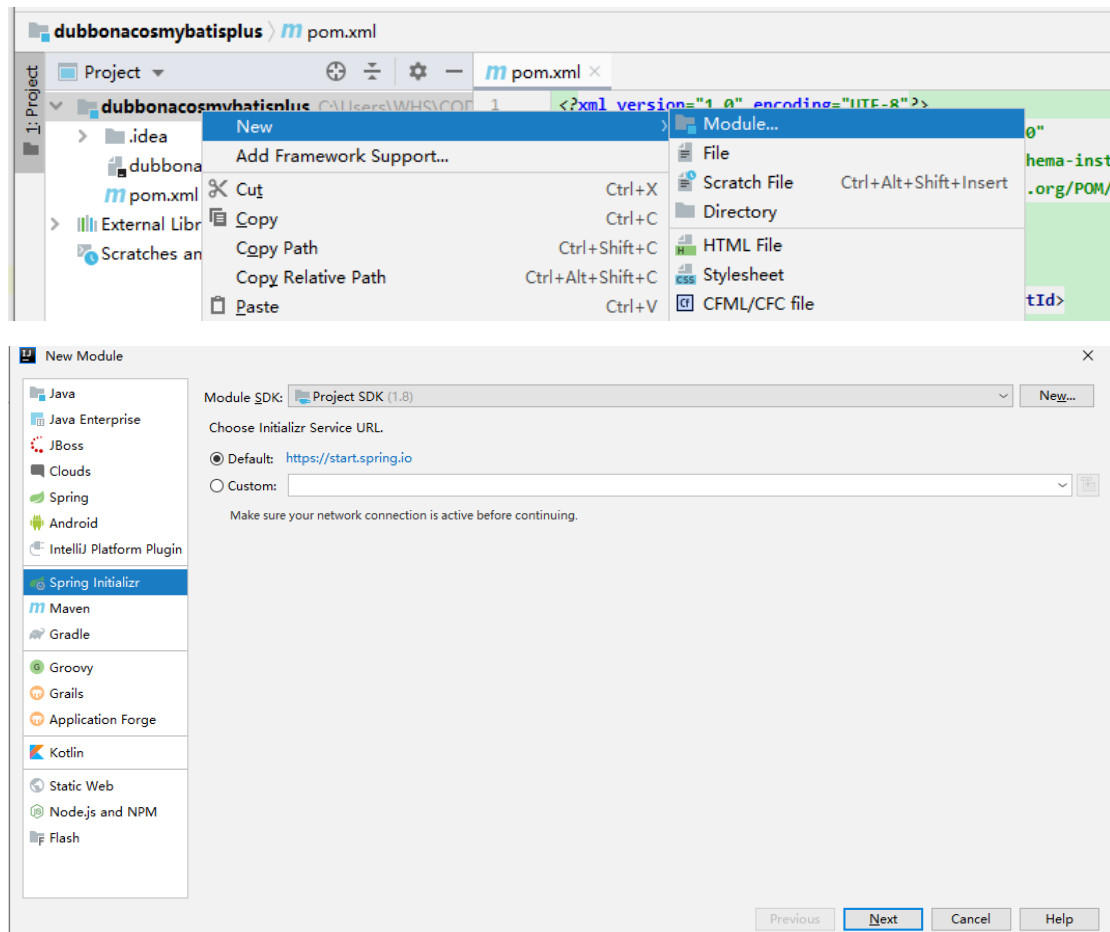
设置工程 pom 文件的打包方式为 pom，如下：



二、创建 dubbo-nacos-mybatisplus-provider 子模块

2.1 创建

右击本工程，new—>Module，如下所示：（创建 springboot 项目）



New Module

Project Metadata

Group: com.whs

Artifact: dubbo-nacos-mybatisplus-provider

Type: Maven Project (Generate a Maven based project archive.)

Language: Java

Packaging: Jar

Java Version: 8

Version: 0.0.1-SNAPSHOT

Name: dubbo-nacos-mybatisplus-provider

Description: Demo project for Spring Boot

Package: com.whs.dubbo-nacos-mybatisplus-provider

Previous Next Cancel Help

New Module

Dependencies

Spring Boot 2.5.2

Selected Dependencies

Select dependencies on the left
Ctrl+F to search in dependencies

Previous Next Cancel Help

New Module

Module name: dubbo-nacos-mybatisplus-provider

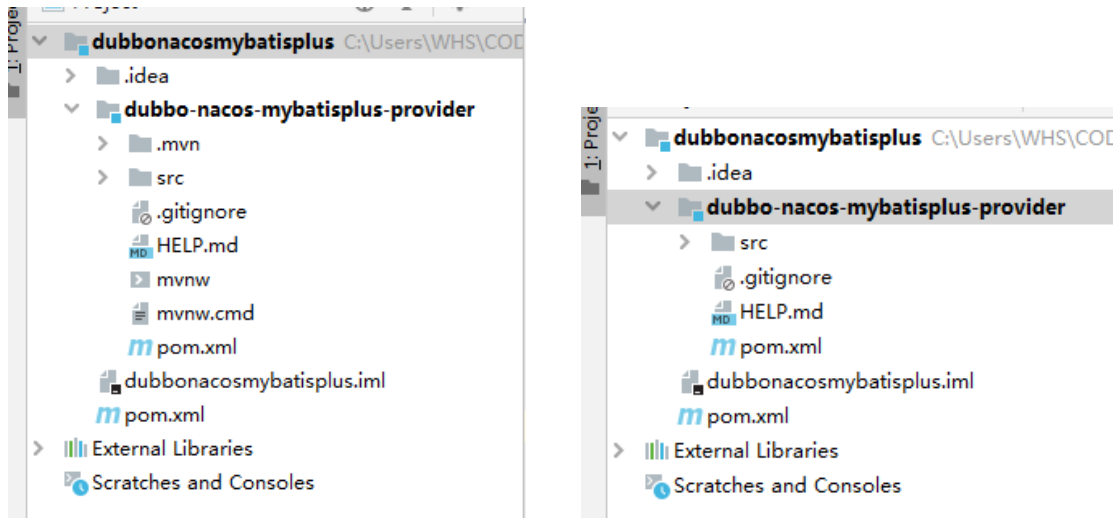
Content root: C:\Users\WHS\CODE\dubbo-nacos-mybatisplus-provider

Module file location: C:\Users\WHS\CODE\dubbo-nacos-mybatisplus-provider

Previous Finish Cancel Help

最后一步，点击 **finish**，完成该模块的创建。

创建该模块后的项目结构如下左图，删除“.mvn”、“mvnw”和“mvnw.cmd”文件，如下右图。

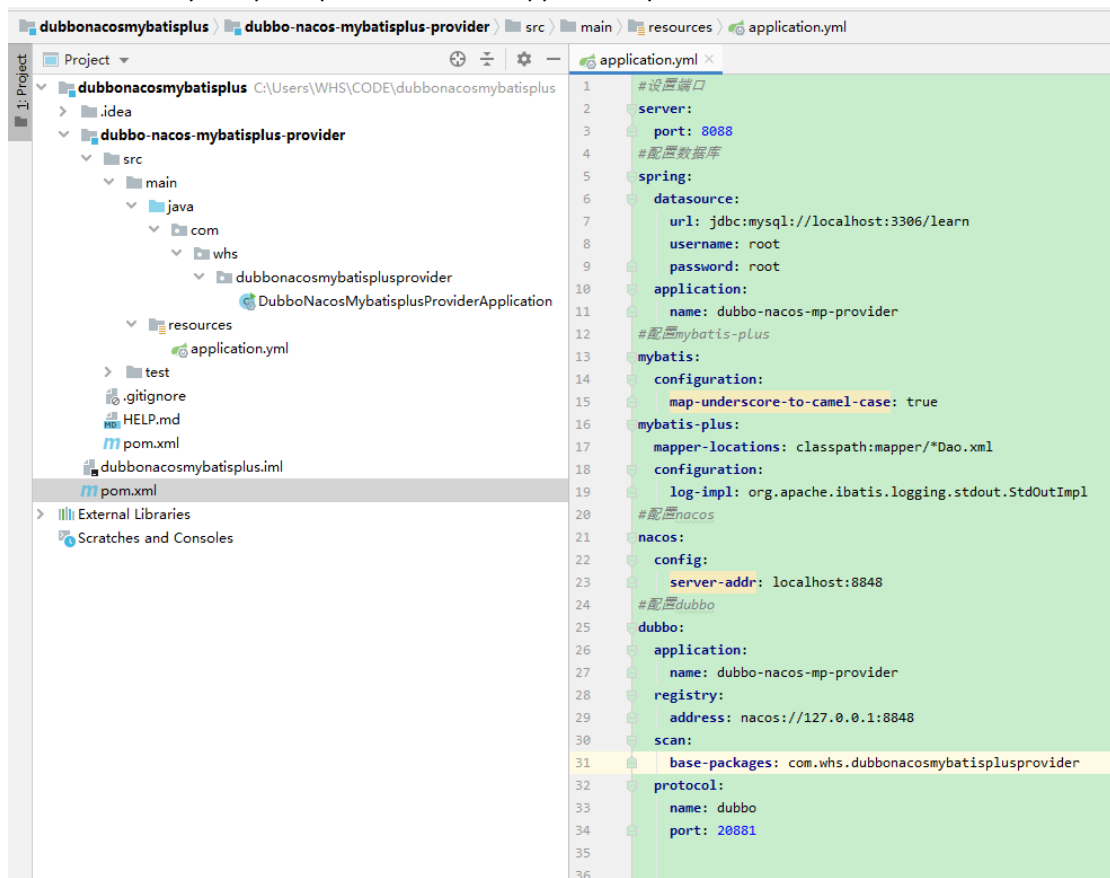


2.2 添加依赖

在 dubbo-nacos-mybatisplus-provider 子模块的 pom 文件中，添加依赖，如[附件 1](#)所示。其中，添加的依赖包括：**mysql**、**mybatis-plus**、**nacos**、**dubbo** 等依赖包。

2.3 application.yml 配置

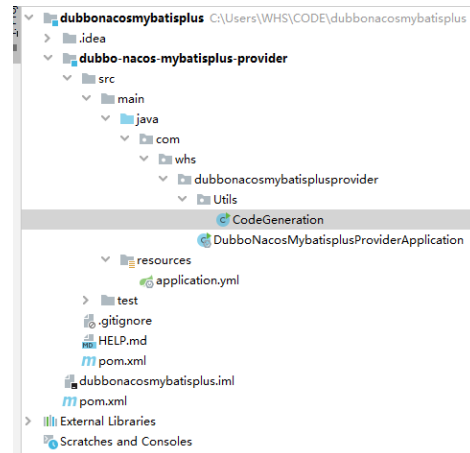
dubbo-nacos-mybatisplus-provider 模块的 application.yml 的配置如下：



2.4 mybatisplus 逆向工程

逆向工程的代码 CodeGeneration 见[附件 2](#)。

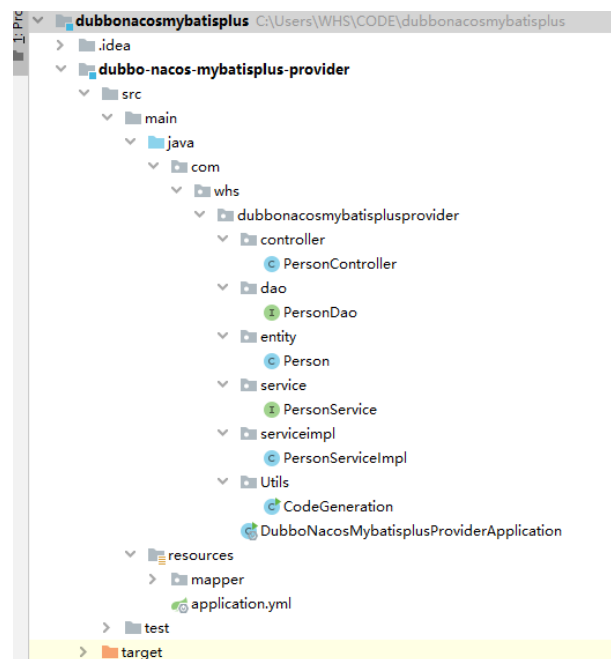
添加逆向生成代码后的模块结构如下：



其中，要注意的是，在生成的 java 的路径中，要加上模块的名称，如下：



在 CodeGeneration 中配置好信息以后，直接运行该类的主函数，最后得到的项目结构如下：

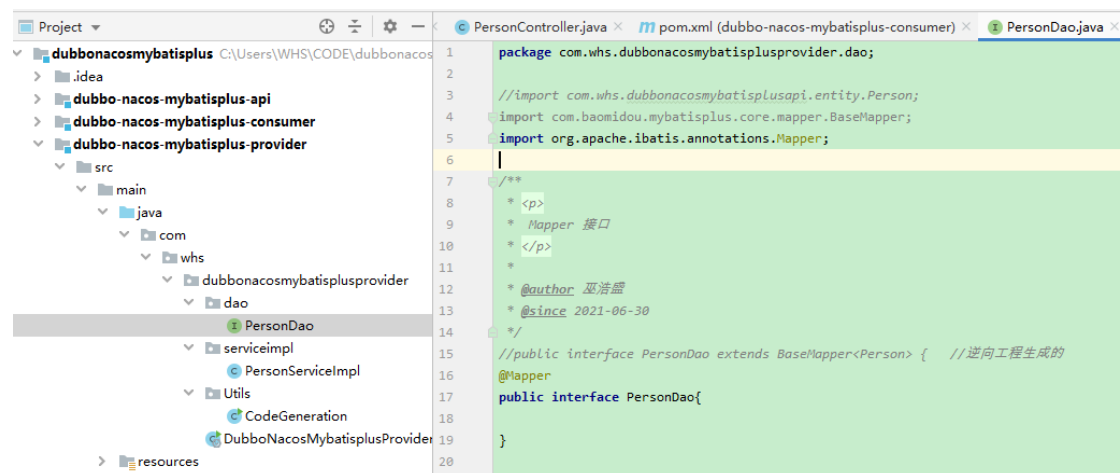


注：接下来会新建 api 模块和 consumer 模块，生成的 entity 和 service 会移到 api 模块，controller 会移到 consumer 模块。

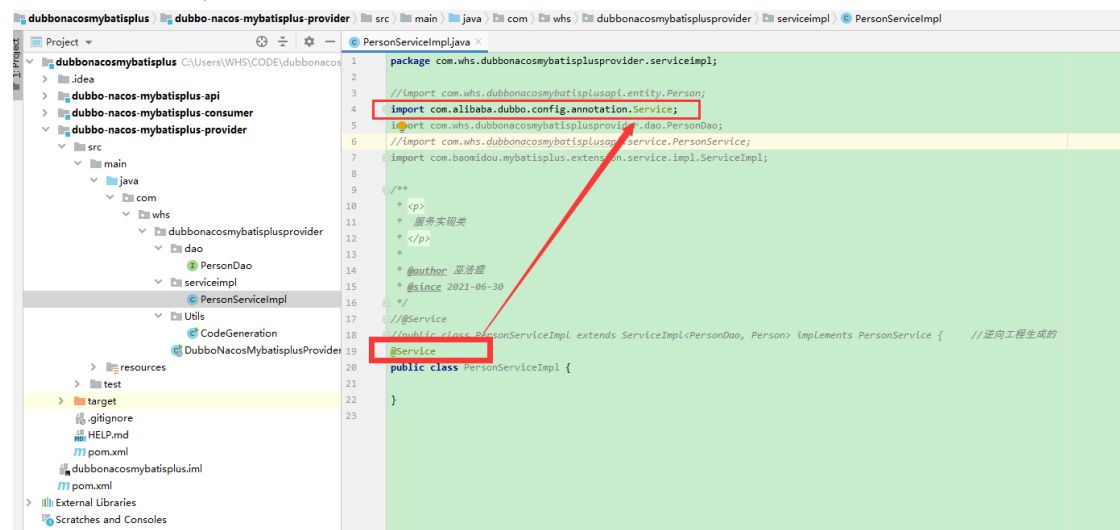
2.5 修改代码

对 PersonDao 和 PersonServiceImpl 进行了改动。具体如下：

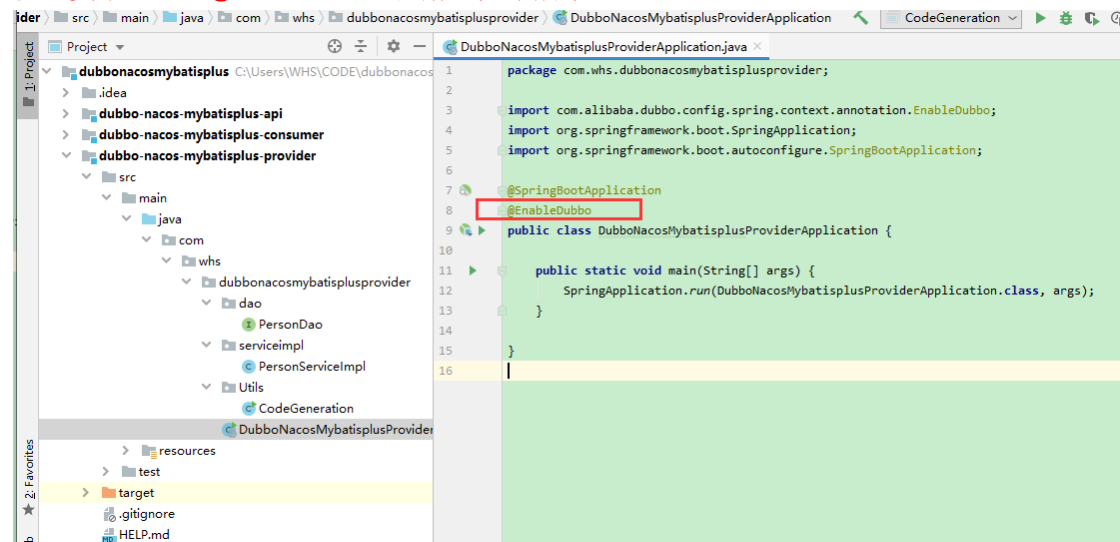
PersonDao 的修改如下，并加上 **@Mapper** 注解（不加会导致启动项目无法扫描到）



PersonServiceImpl 修改如下，并将 **@Service** 换成 dubbo 的 **@Service** 注解。



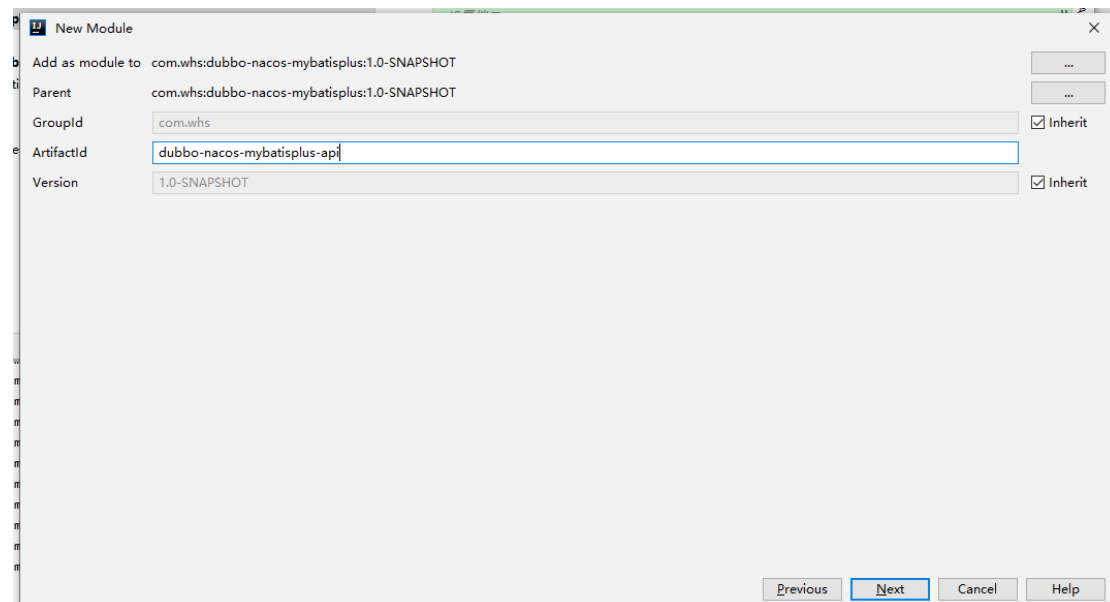
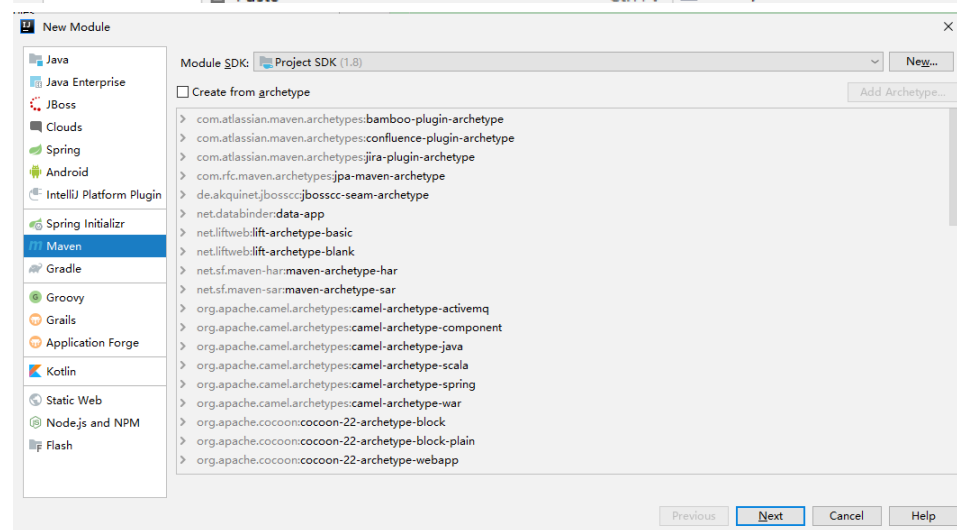
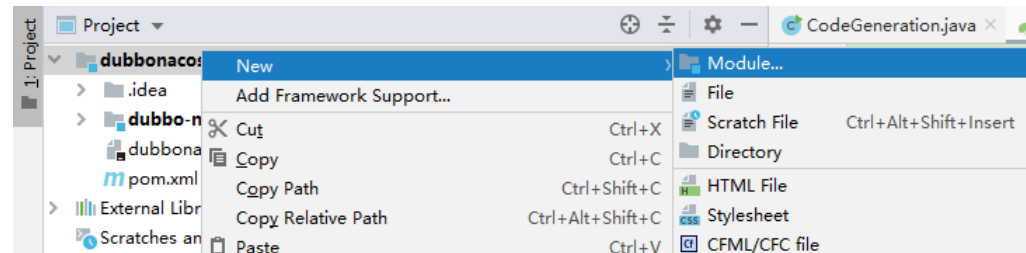
在主类中，加上 **@EnableDubbo** 注解，如下所示：

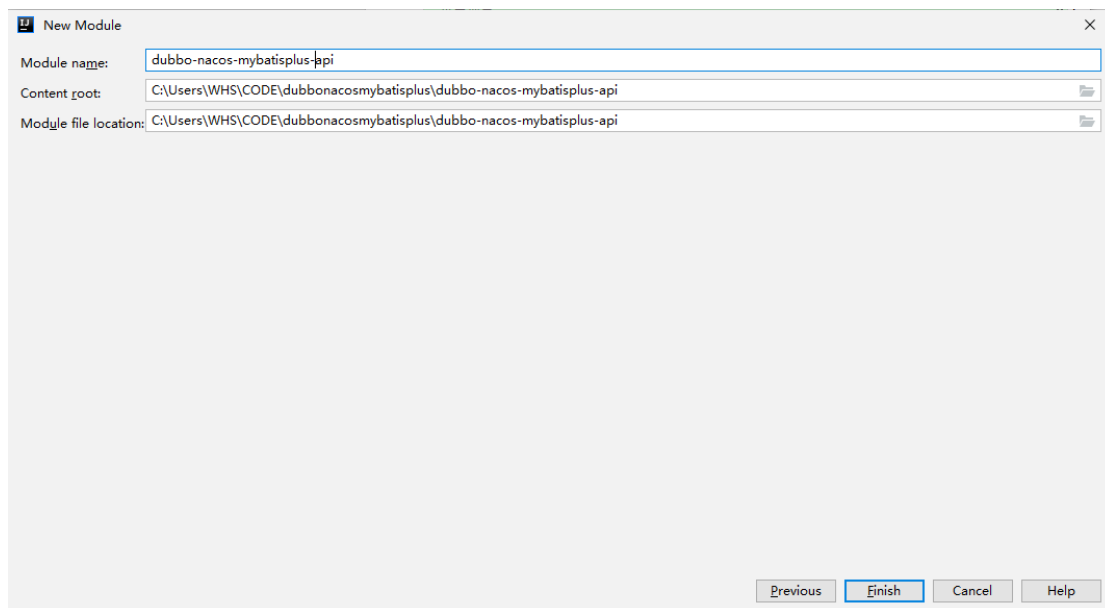


三、创建 dubbo-nacos-mybatisplus-api 子模块

3.1 创建

右击本工程，new—>Module，如下所示：（创建 maven 项目）

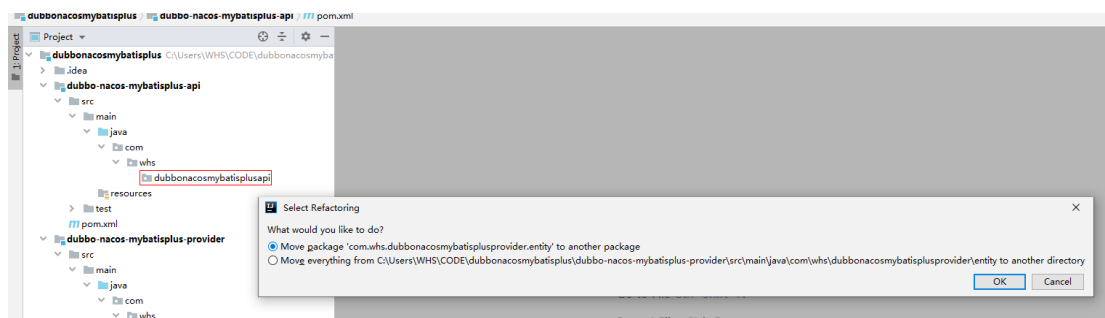




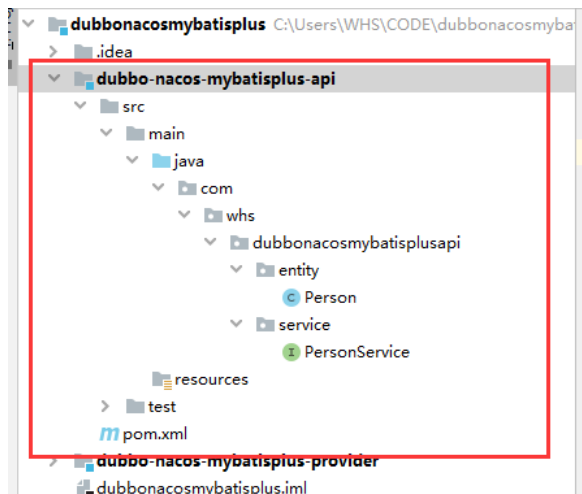
点击 finish 完成 dubbo-nacos-mybatisplus-api 子模块的创建。

3.2 添加 java 代码

将 dubbo-nacos-mybatisplus-provider 模块逆向工程生成的 entity 和 service 代码移动到 dubbo-nacos-mybatisplus-api 模块下，如下所示。

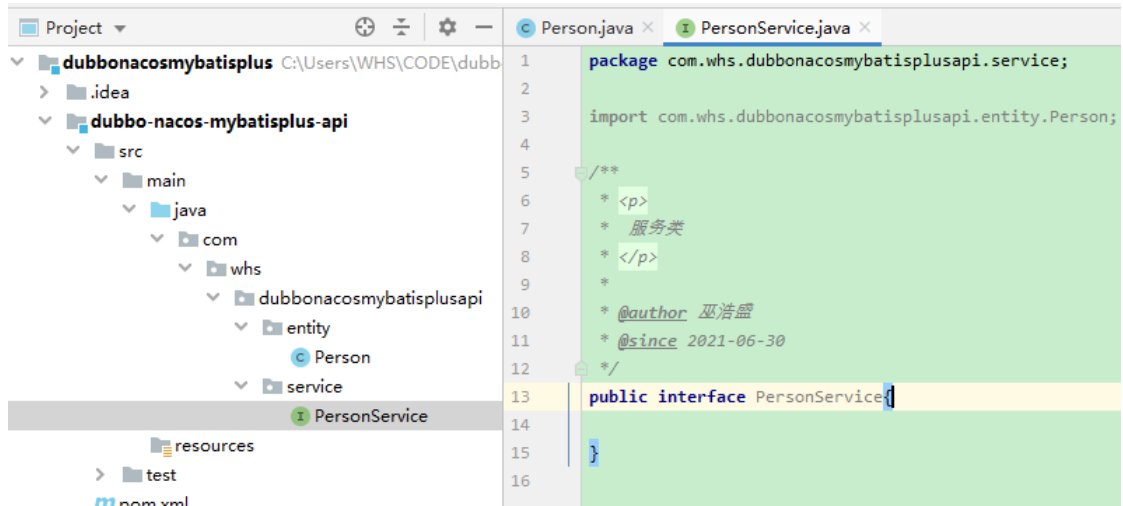
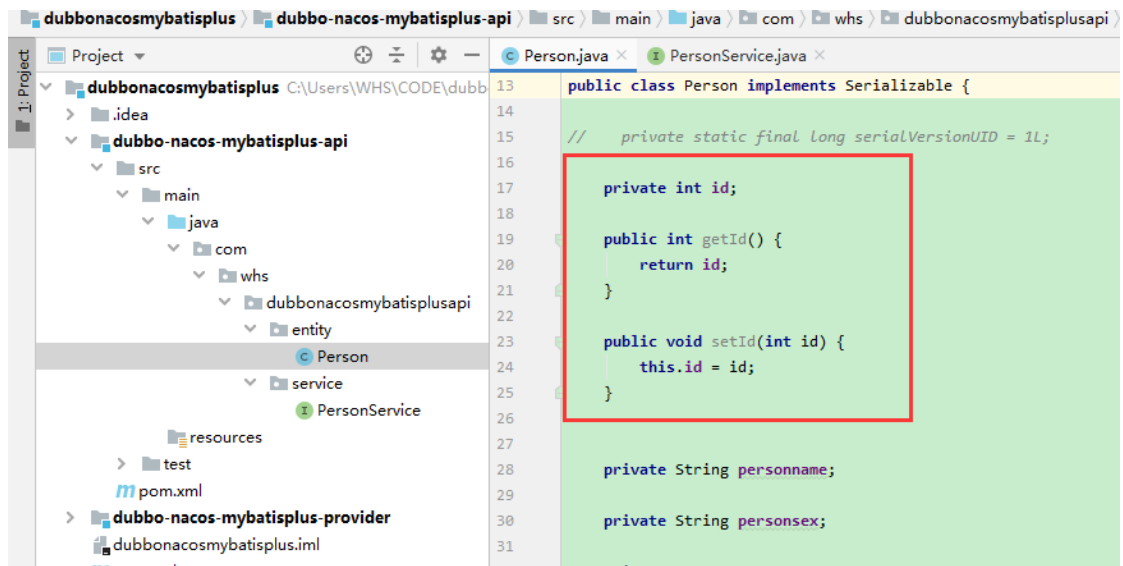


该模块结构如下图所示：



此处改动了 Person 实体和 PersonService 的代码，分别下图。

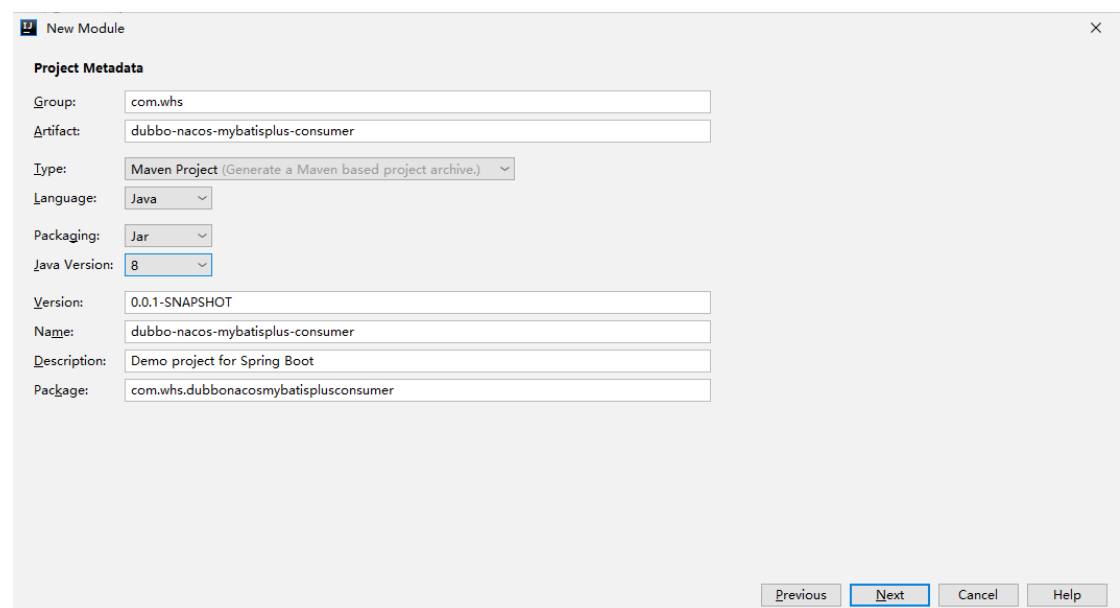
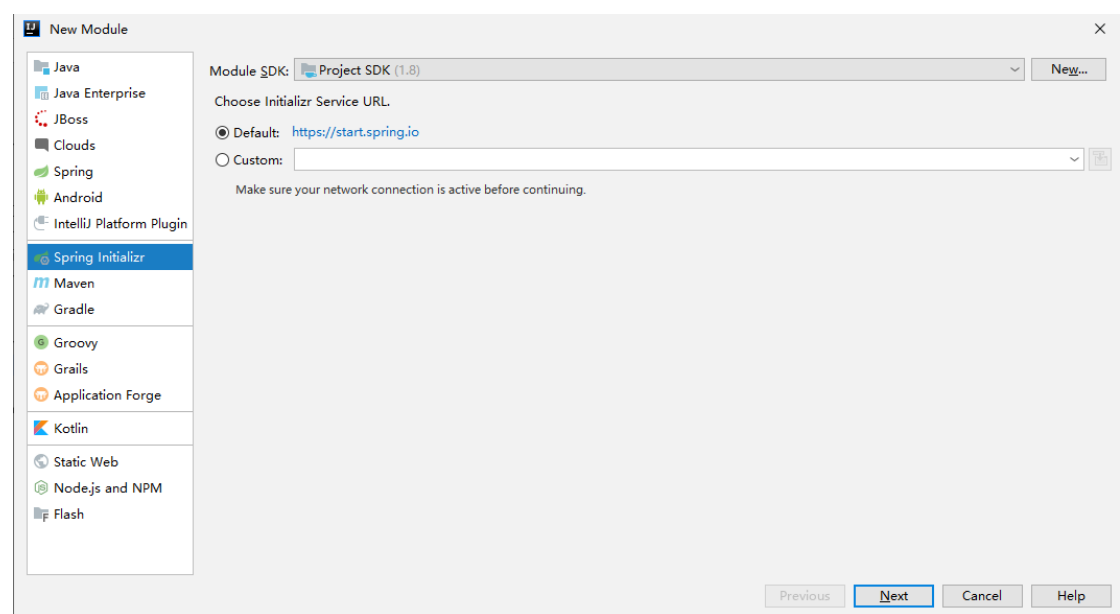
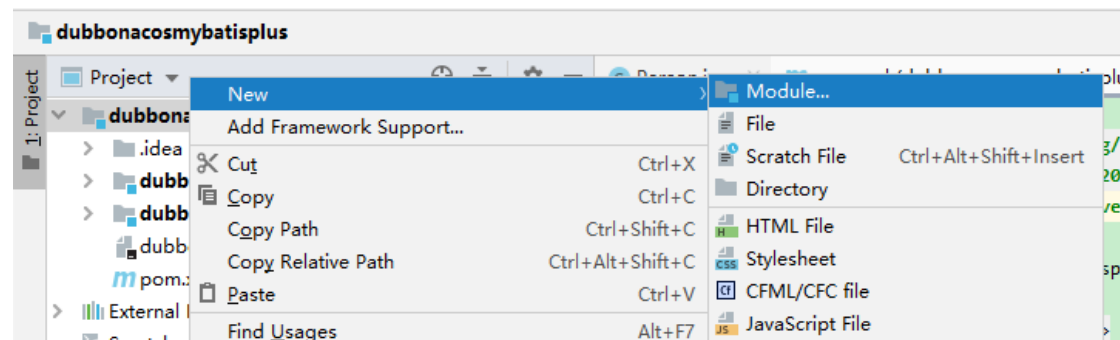
Person 这个类注释掉了 serialVersionUID，自己加入了 id 以及 set 和 get 方法。

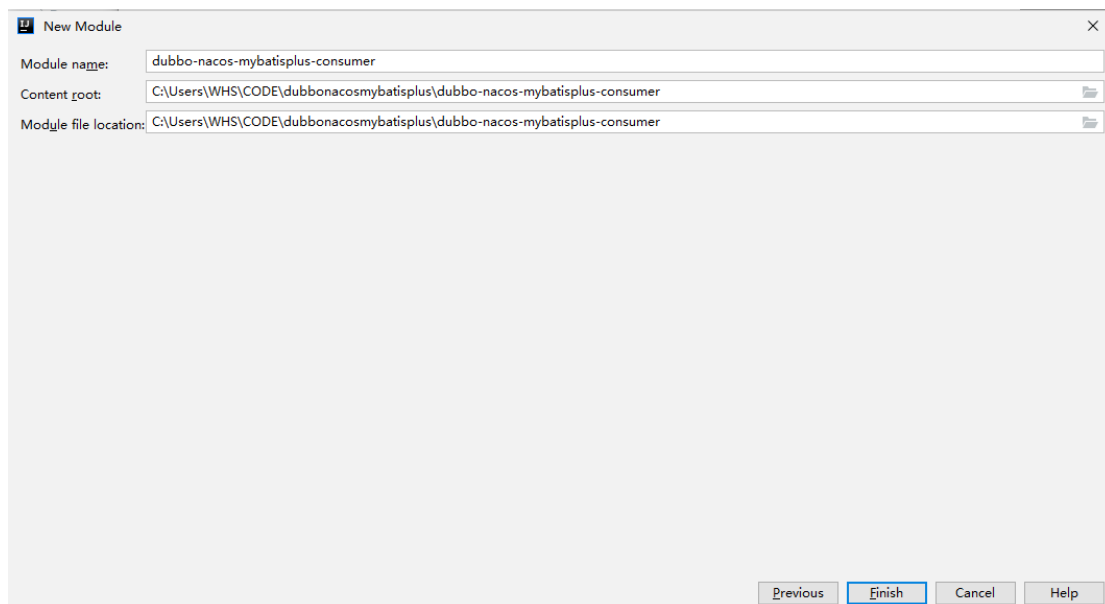
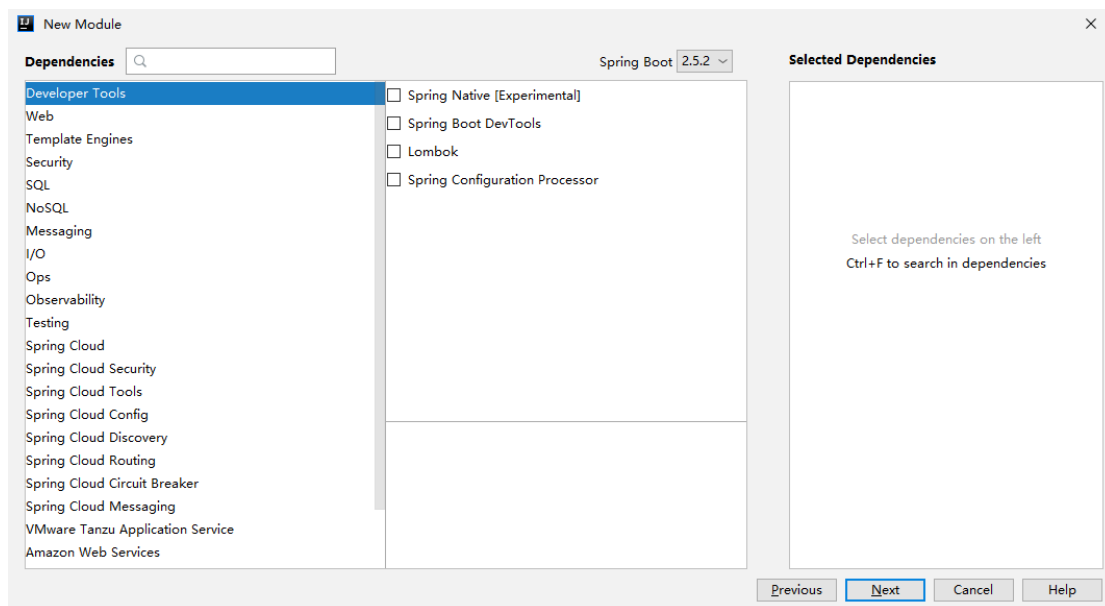


四、创建 dubbo-nacos-mybatisplus-consumer 子模块

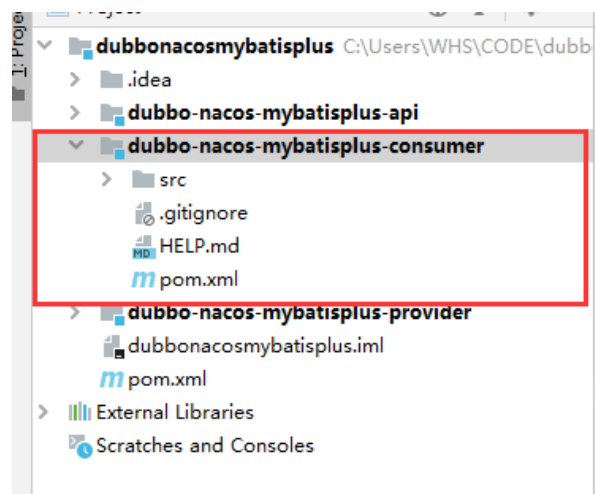
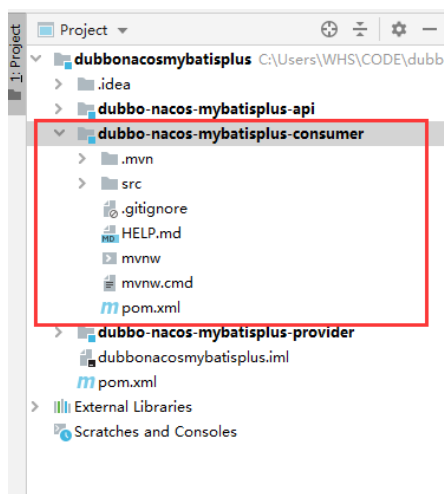
4.1 创建

右击本工程，new—>Module，如下所示：（创建 **springboot** 项目）





创建该模块后的项目结构如下左图，删除 “.mvn”、“mvnw” 和 “mvnw.cmd” 文件，如下右图。



4.2 添加依赖

在 dubbo-nacos-mybatisplus-consumer 子模块的 pom 文件中，添加依赖，所有依赖如下。

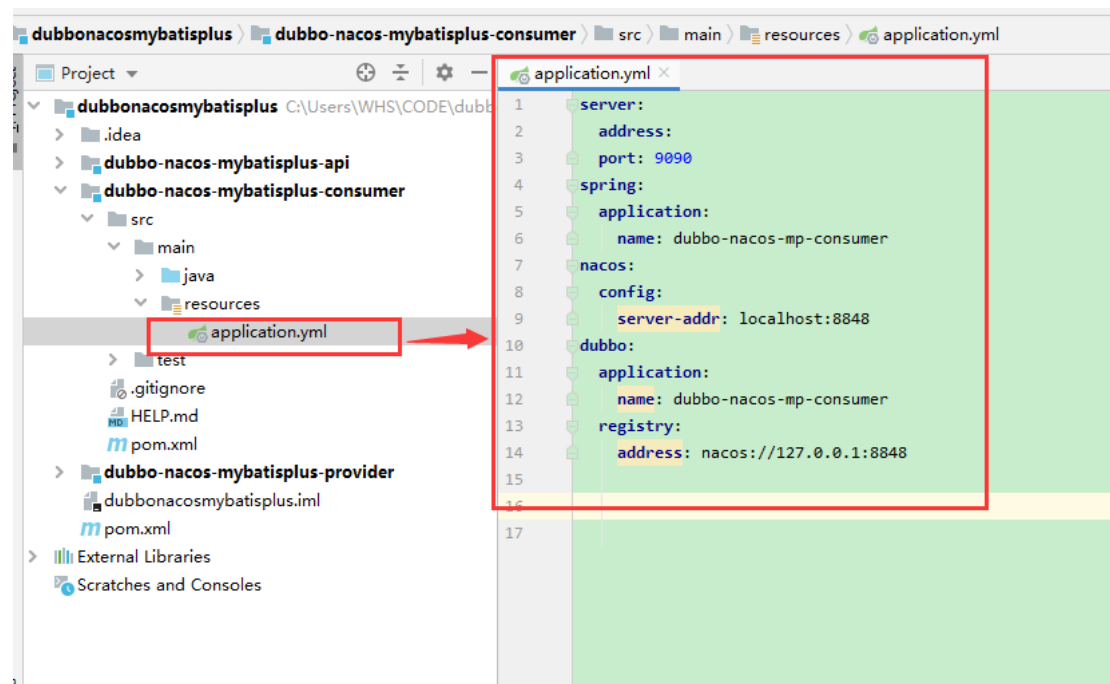
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- Dubbo Nacos registry dependency -->
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>dubbo-registry-nacos</artifactId>
    <version>0.0.1</version>
  </dependency>
  <!-- Dubbo dependency -->
  <dependency>
    <groupId>com.alibaba.boot</groupId>
    <artifactId>dubbo-spring-boot-starter</artifactId>
    <version>0.2.0</version>
  </dependency>

  <dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>2.12.0</version>
  </dependency>
</dependencies>
```

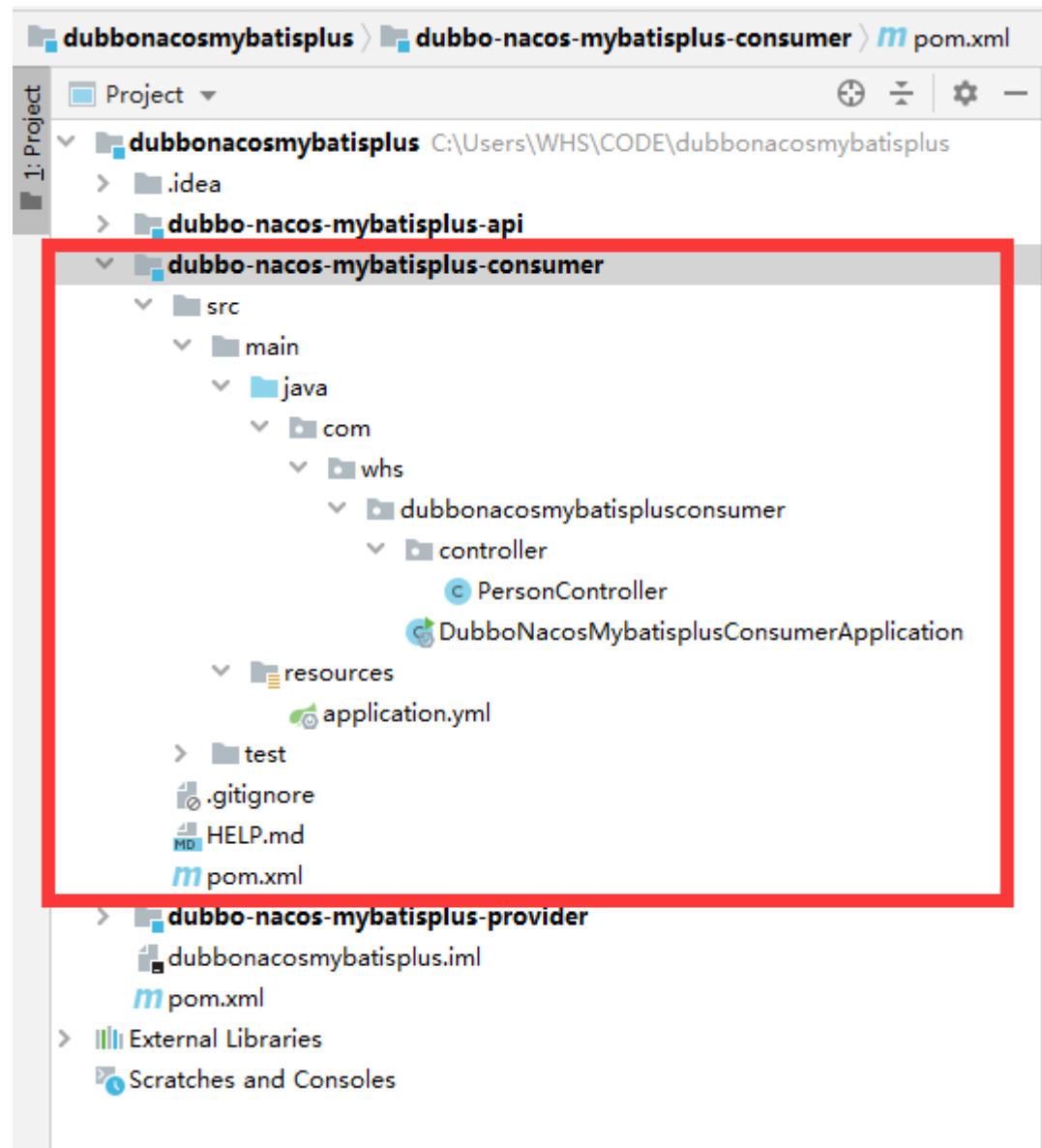
4.3 application.yml 配置

dubbo-nacos-mybatisplus-consumer 模块的 application.yml 的配置如下：



4.4 添加 java 代码

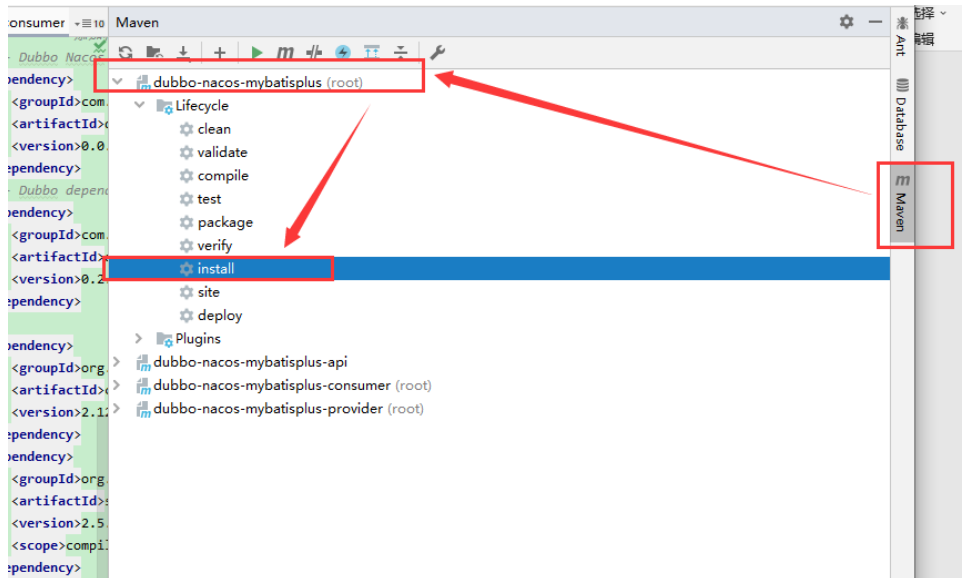
将 dubbo-nacos-mybatisplus-provider 模块逆向工程生成的 controller 代码移动到 dubbo-nacos-mybatisplus-consumer 模块下,操作与 dubbo-nacos-mybatisplus-api 模块添加 java 代码的操作一样, 最后结构如下所示。



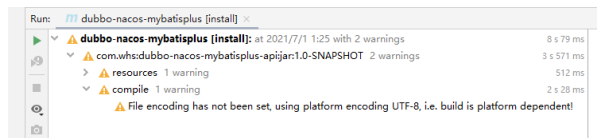
五、模块之间的依赖配置

5.1 模块打包

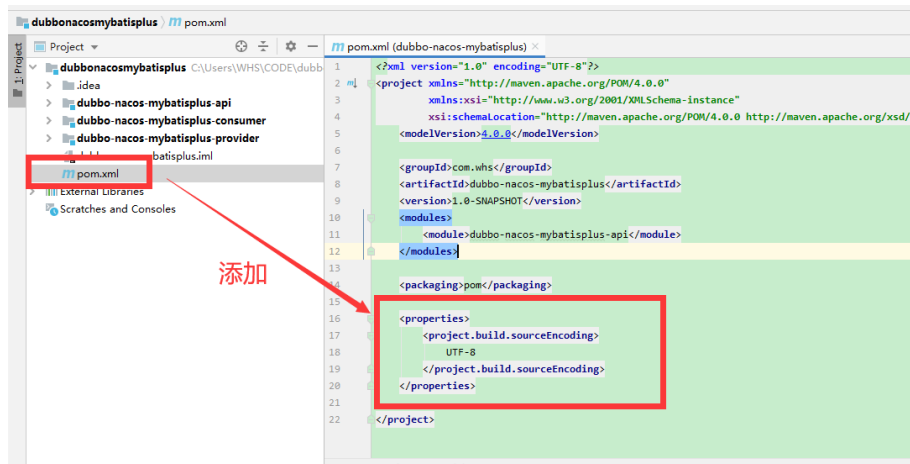
如下所示，直接对整个工程打包。最后双击 `install` 等待打包完成。



其中，可能会遇到以下警告（如果没遇到一下问题则不用理会，忽略以下步骤即可）。

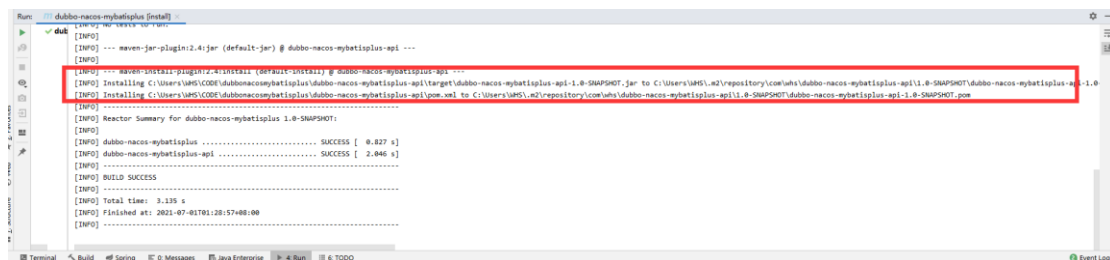


直接在本工程的 `pom` 文件添加以下即可，如下：



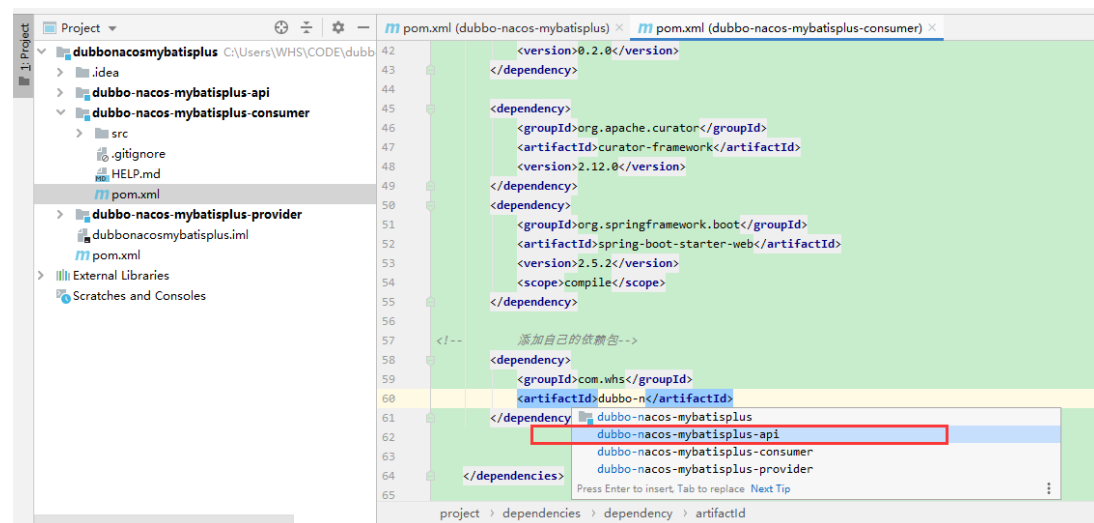
重新 `install` 即可。

如下所示，可以看到打包的信息，以及打包之后，文件所在的位置。

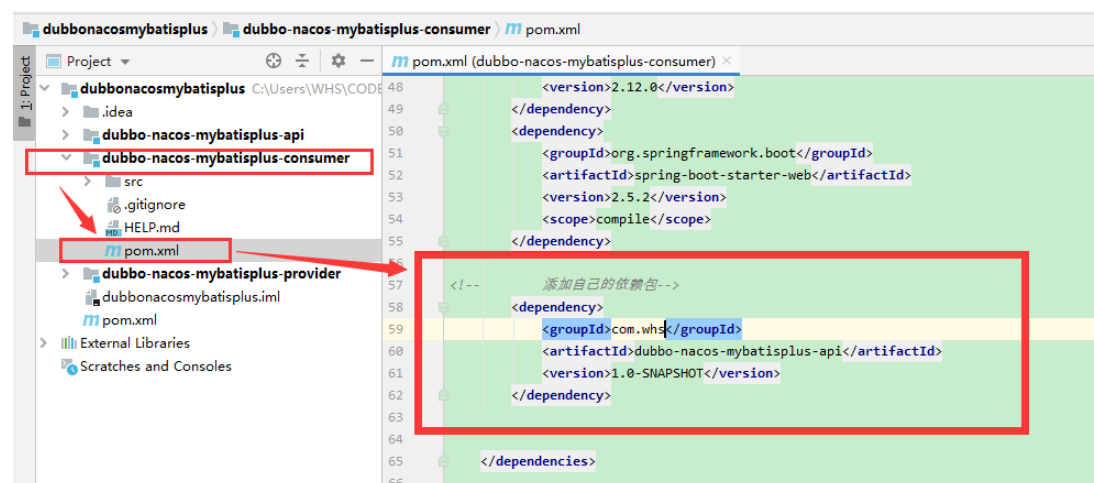


5.2 添加依赖

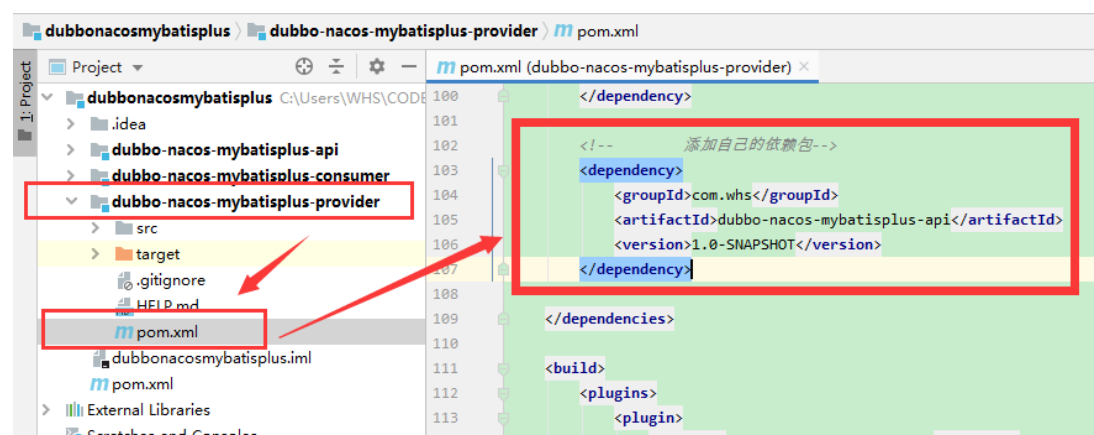
这一步是为我们自己的写的模块添加自己写的接口的依赖包。



可以看到给出了提示，表明自己写的东西已经被打包好，添加即可。如下所示，是在 dubbo-nacos-mybatisplus-consumer 模块中的 pom 文件添加依赖。

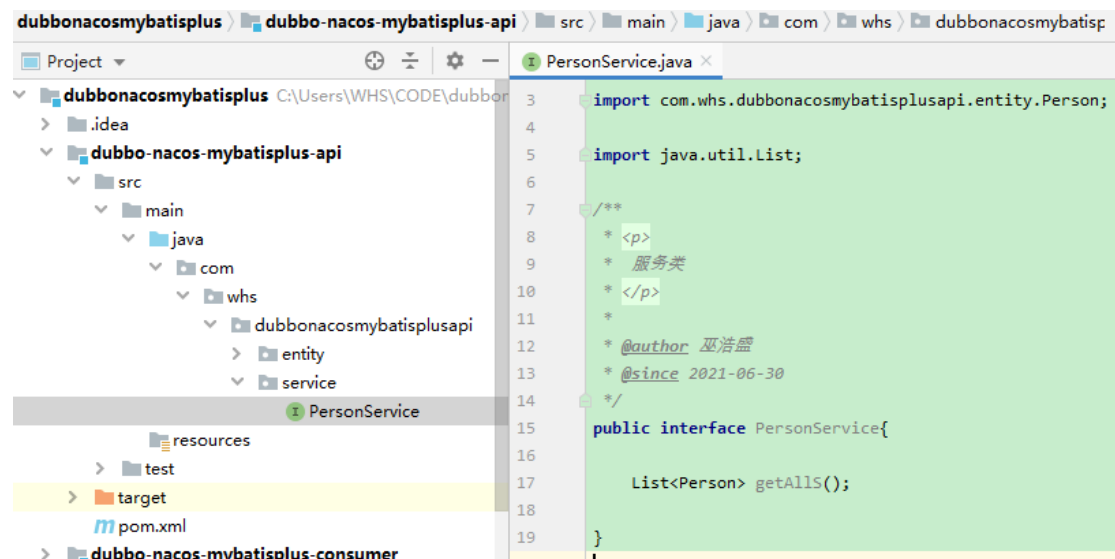


对 dubbo-nacos-mybatisplus-provider 模块的 pom 文件添加同样的依赖包。如下所示：



六、添加代码

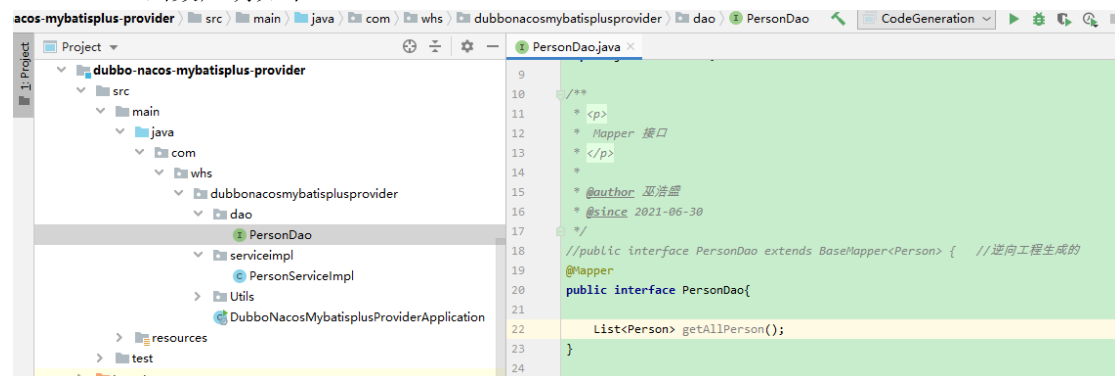
6.1 dubbo-nacos-mybatisplusr-api 接口声明函数



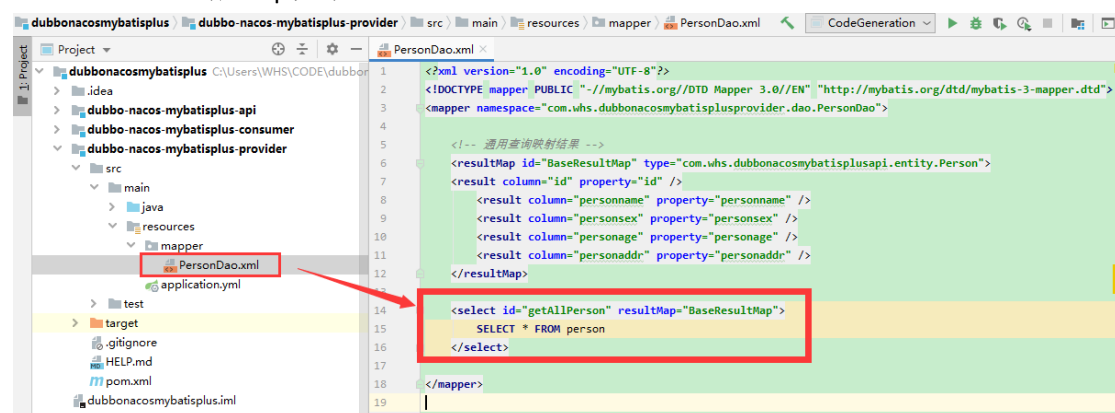
6.2 dubbo-nacos-mybatisplusr-provider 函数实现

6.2.1 PersonDao 函数声明以及 PersonDao.xml 的 sql

PersonDao 函数声明如下：

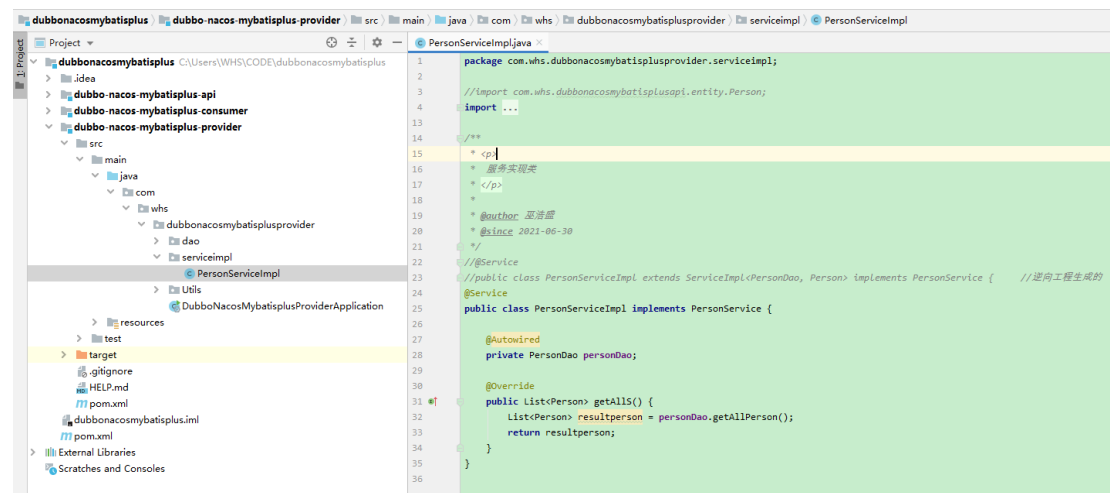


PersonDao.xml 对应 sql 如下：



6.2.2 PersonServiceImpl 接口函数实现

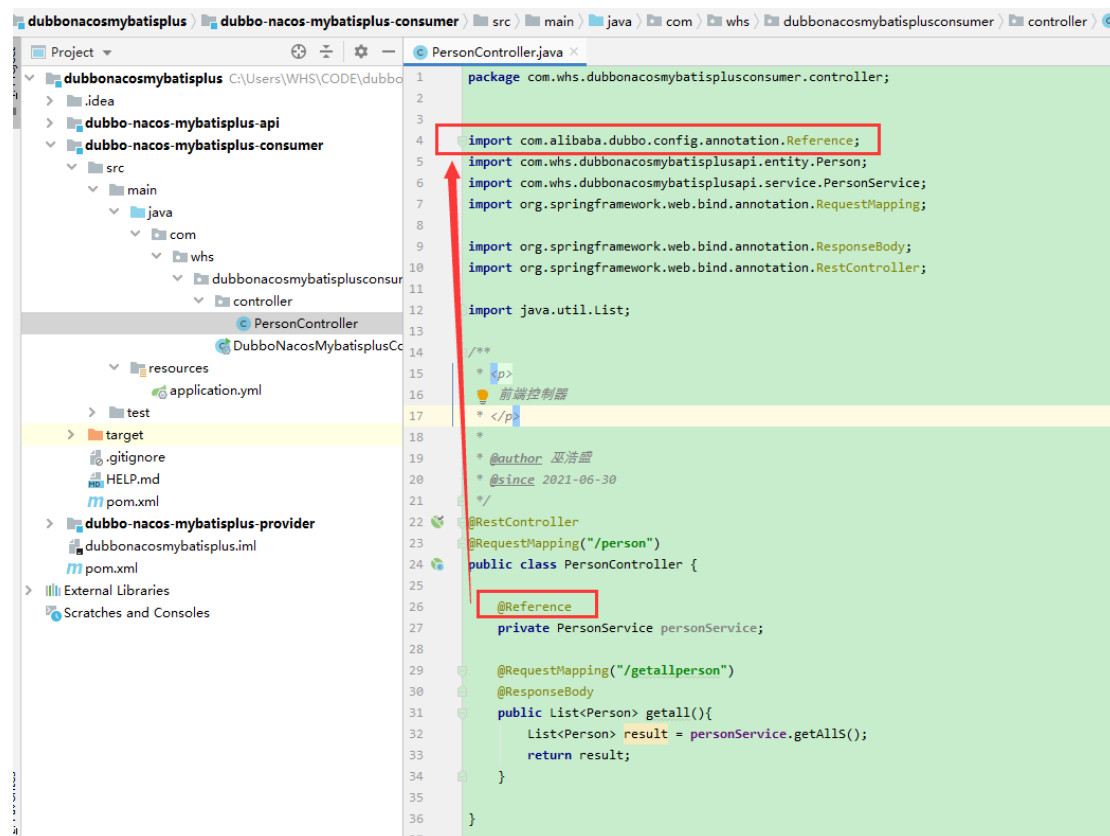
PersonServiceImpl 的实现如下：



```
1 package com.whs.dubbo.nacos.mybatis.plus.provider.serviceimpl;
2
3 //import com.whs.dubbo.nacos.mybatis.plus.api.entity.Person;
4 import ...
5
6 /**
7  * 服务实现类
8  */
9
10 * @author 巫浩磊
11 * @since 2021-06-30
12 */
13
14 //Service
15 //public class PersonServiceImpl extends ServiceImpl<PersonDao, Person> implements PersonService { //逆向工程生成的
16 @Service
17 public class PersonServiceImpl implements PersonService {
18
19     @Autowired
20     private PersonDao personDao;
21
22     @Override
23     public List<Person> getAll() {
24         List<Person> resultPerson = personDao.getAllPerson();
25         return resultPerson;
26     }
27 }
28
29
30
31
32
33
34
35
36
```

6.3 dubbo-nacos-mybatisplusr-consumer 调用

PersonController 的实现如下：



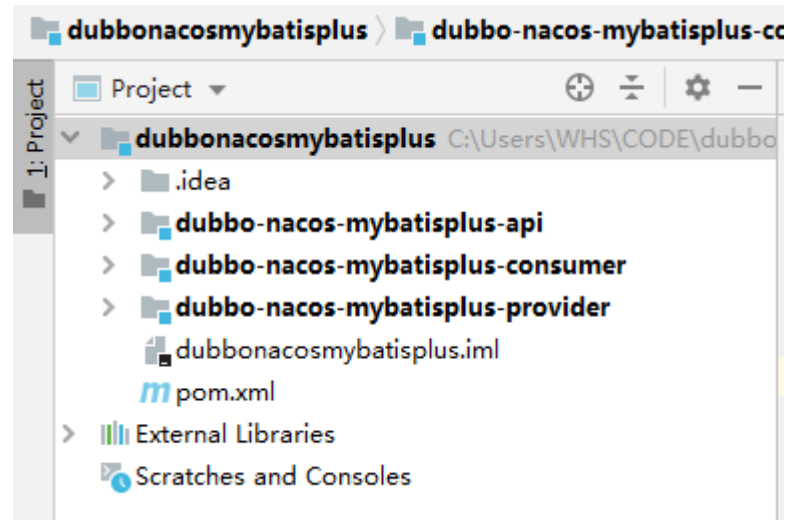
```
1 package com.whs.dubbo.nacos.mybatis.plus.consumer.controller;
2
3
4 import com.alibaba.dubbo.config.annotation.Reference;
5 import com.whs.dubbo.nacos.mybatis.plus.api.entity.Person;
6 import com.whs.dubbo.nacos.mybatis.plus.api.service.PersonService;
7 import org.springframework.web.bind.annotation.RequestMapping;
8
9 import org.springframework.web.bind.annotation.ResponseBody;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import java.util.List;
13
14 /**
15  *
16  * 前端控制器
17  */
18
19 * @author 巫浩磊
20 * @since 2021-06-30
21 */
22
23 @RestController
24 @RequestMapping("/person")
25 public class PersonController {
26
27     @Reference
28     private PersonService personService;
29
30     @RequestMapping("/getallperson")
31     @ResponseBody
32     public List<Person> getall(){
33         List<Person> result = personService.getAll();
34         return result;
35     }
36 }
37
```

注意：此处的@Reference注解需要使用 dubbo 提供的。

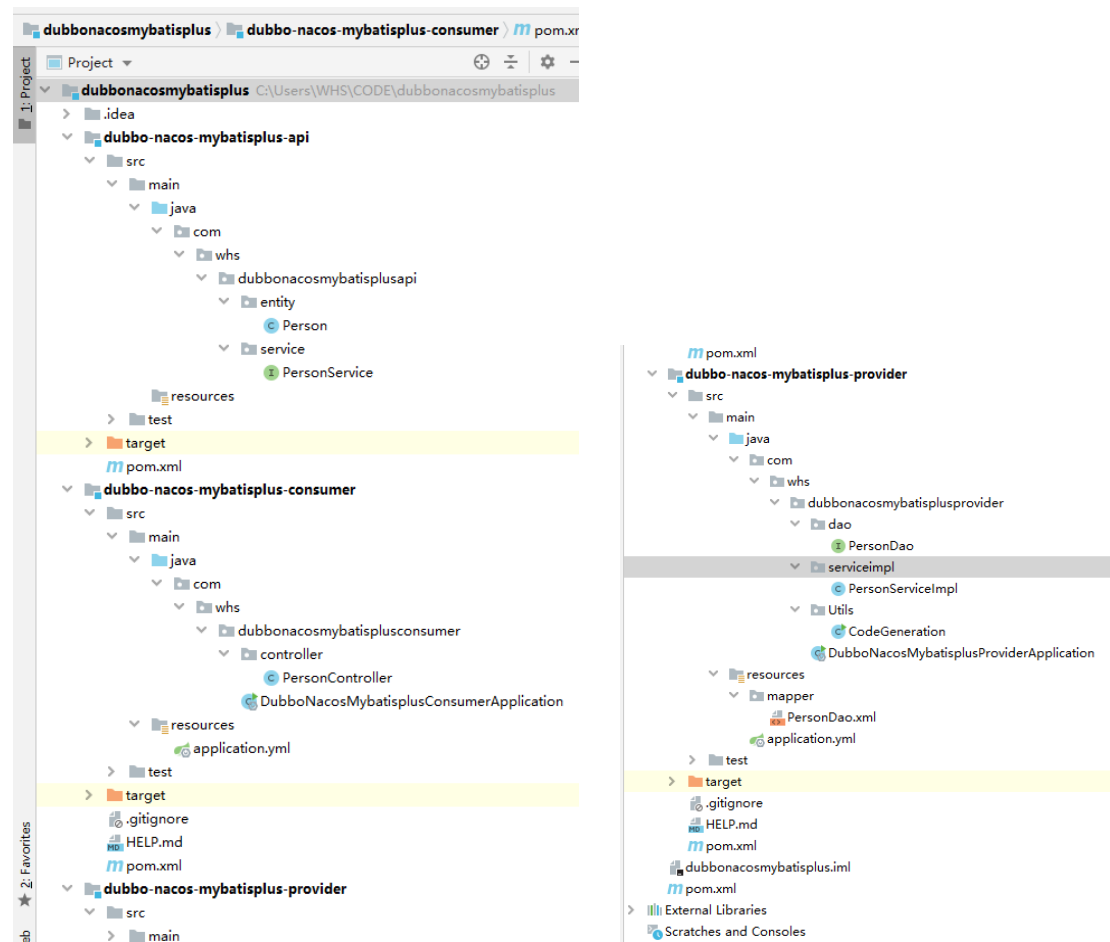
项目结构

至此，项目就搭建完成，最终的项目结构如下所示：

项目总结构：



项目详细结构：



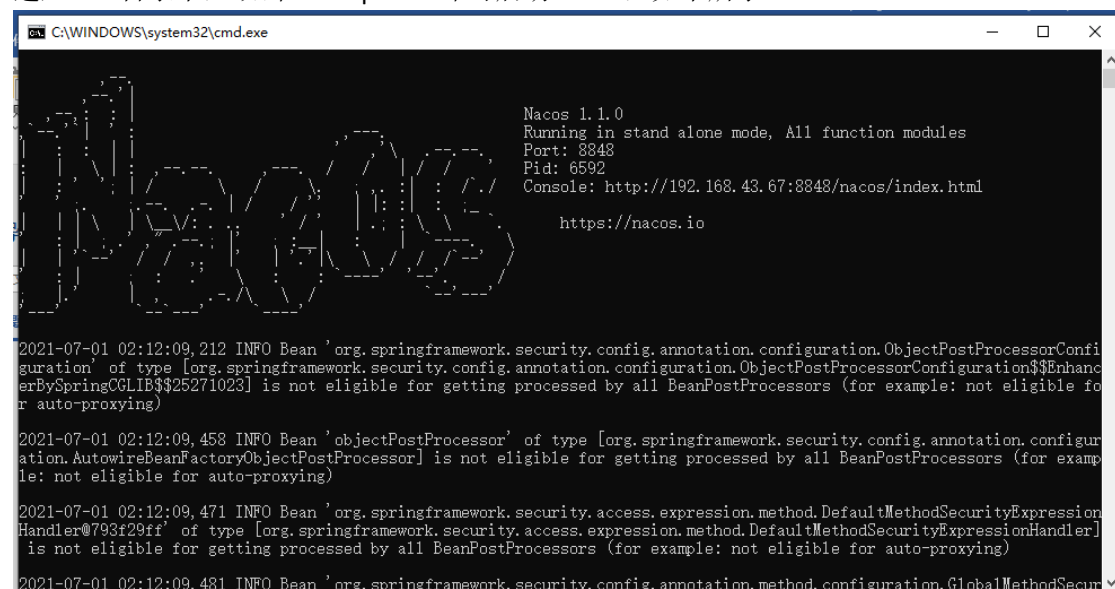
项目运行

1. 启动 nacos

下载 nacos 的压缩包，解压得到以下文件：

↑ > WHS > CODE > nacos文件 > nacos-server-1.1.0 > nacos >					
名称	修改日期	类型	大小		
bin	2021/6/28 19:56	文件夹			
conf	2021/6/28 17:46	文件夹			
data	2021/6/28 19:56	文件夹			
logs	2021/7/1 1:58	文件夹			
plugins	2021/6/28 17:46	文件夹			
target	2021/6/28 17:46	文件夹			
LICENSE	2019/5/13 10:33	文件	17 KB		
NOTICE	2019/5/13 10:33	文件	2 KB		

进入 bin 目录下，双击 startup.cmd 即可启动 nacos，如下所示：



```
C:\WINDOWS\system32\cmd.exe

Nacos 1.1.0
Running in stand alone mode, All function modules
Port: 8848
Pid: 6592
Console: http://192.168.43.67:8848/nacos/index.html
https://nacos.io

2021-07-01 02:12:09,212 INFO Bean 'org.springframework.security.config.annotation.configuration.ObjectPostProcessorConfiguration' of type [org.springframework.security.config.annotation.configuration.ObjectPostProcessorConfiguration$$EnhancerBySpringCGLIB$$25271023] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2021-07-01 02:12:09,458 INFO Bean 'objectPostProcessor' of type [org.springframework.security.config.annotation.configuration.AutowireBeanFactoryObjectPostProcessor] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2021-07-01 02:12:09,471 INFO Bean 'org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler@793f29ff' of type [org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2021-07-01 02:12:09,481 INFO Bean 'org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration' of type [org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
```

2. 运行启动类

依次运行 dubbo-nacos-mybatisplus-provider 和 dubbo-nacos-mybatisplus-consumer 模块的主类。

3. 测试

在浏览器中输入如下：<http://localhost:9090/person/getallperson>，得到如下：



其中，nacos 后台管理的服务列表如下：（在浏览器中输入 <http://localhost:8848/nacos> 即可，然后输入用户名和密码，均为 nacos）

NACOS

首页 文档 博客 社区 En nacos

NACOS 1.1.0

配置管理

配置列表

历史版本

监听查询

服务管理

服务列表

订阅者列表

命名空间

集群管理

节点列表

public

服务列表 | public

服务名称

服务名	分组	集群数目	实例数	健康实例数	操作
providers:com.whs.dubboacosmybatisplusapi.service.PersonService	DEFAULT_GROUP	1	1	1	详情 示例代码 删除
consumers:com.whs.dubboacosmybatisplusapi.service.PersonService	DEFAULT_GROUP	1	1	1	详情 示例代码 删除

附件

附件 1 dubbo-nacos-mybatisplusr-provider 子模块的依赖

dubbo-nacos-mybatisplusr-provider 子模块的 pom 文件中需要添加的依赖。

```
<!-- mysql-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<!-- 这是 Lombok 的依赖 -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<!-- 这是 mybatis-plus 依赖 -->
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>3.1.1</version>
</dependency>
<!-- 这是 mybatis-plus 的代码自动生成器 -->
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-generator</artifactId>
  <version>3.1.1</version>
</dependency>
<!-- 这是模板引擎依赖 -->
<dependency>
  <groupId>org.freemarker</groupId>
  <artifactId>freemarker</artifactId>
  <version>2.3.28</version>
</dependency>
<!-- 添加相应依赖-->
<dependency>
  <groupId>com.alibaba.nacos</groupId>
  <artifactId>nacos-client</artifactId>
  <version>1.0.1</version>
</dependency>
<dependency>
  <groupId>com.alibaba.spring</groupId>
  <artifactId>spring-context-support</artifactId>
  <version>1.0.2</version>
</dependency>
<dependency>
  <groupId>io.netty</groupId>
  <artifactId>netty-all</artifactId>
  <version>4.1.32.Final</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>dubbo-registry-nacos</artifactId>
  <version>2.6.7</version>
</dependency>
<dependency>
  <groupId>com.alibaba.boot</groupId>
  <artifactId>dubbo-spring-boot-starter</artifactId>
  <version>0.2.0</version>
</dependency>
<dependency>
  <groupId>org.apache.curator</groupId>
  <artifactId>curator-framework</artifactId>
  <version>2.12.0</version>
</dependency>
```

附件 2 CodeGeneration 代码

CodeGeneration 的代码如下:

```
public class CodeGeneration {
    //根据表生成相关的 java 代码
    public static void main(String[] args) {
        AutoGenerator mpg = new AutoGenerator();
        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir");
        gc.setOutputDir(projectPath + "/dubbo-nacos-mybatisplus-provider/src/main/java");//输出
        // 文件路径
        System.out.println("projectpath:"+projectPath);
        gc.setOpen(false);
        gc.setFileOverride(true);
        gc.setActiveRecord(false); // 不需要 ActiveRecord 特性的请改为 false
        gc.setEnableCache(false); // XML 二级缓存
        gc.setBaseResultMap(true); // XML resultMap
        gc.setBaseColumnList(false); // XML columnList
        gc.setAuthor("巫浩盛"); //设置作者
        gc.setDateType(DateType.ONLY_DATE); //设置生成时间
        // 自定义文件命名, 注意 %s 会自动填充表实体属性!
        gc.setServiceName("%sService");
        gc.setMapperName("%sDao");
        mpg.setGlobalConfig(gc);
        // 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://localhost:3306/learn?useUnicode=true&useSSL=false&characterEncoding=utf8&serverTimezone=UTC");
        dsc.setDriverName("com.mysql.cj.jdbc.Driver");
        dsc.setUsername("root");
        dsc.setPassword("root");
        mpg.setDataSource(dsc);
        PackageConfig pc = new PackageConfig();
        //设置文件的生成的包的位置: 修改为自己的项目路径
        pc.setParent("com.whs.dubbonacosmybatisplusprovider");
        pc.setController("controller");
        pc.setService("service");
        pc.setServiceImpl("serviceimpl");
        pc.setMapper("dao");
        pc.setEntity("entity");
        mpg.setPackageInfo(pc);
        // 自定义配置
        InjectionConfig cfg = new InjectionConfig() {
            @Override
            public void initMap() {
                // to do nothing
            }
        };
        List<FileOutConfig> focList = new ArrayList<>();
        //下面的代码需要用到 freemarker 的相关依赖
        focList.add(new FileOutConfig("/templates/mapper.xml.ftl") {
            @Override
            public String outputFile(TableInfo tableInfo) {
                return projectPath +
                //dubbo-nacos-mybatisplus-provider/src/main/resources/mapper/" + "/" + tableInfo.getEntityName()
                + "Dao" + StringPool.DOT_XML; //mapper.xml 文件路径
            }
        });
        cfg.setFileOutConfigList(focList);
        mpg.setCfg(cfg);
        mpg.setTemplate(new TemplateConfig().setXml(null));
        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);
        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        //strategy.setEntityLombokModel(true); //是否系列化类
        strategy.setRestControllerStyle(true);
        strategy.setInclude(new String[] { "person" }); //设置要生成的代码的表名
        strategy.setSuperEntityColumns("id");
        strategy.setControllerMappingHyphenStyle(true);
        strategy.setTablePrefix(pc.getModuleName() + "_");
        mpg.setStrategy(strategy);
        mpg.setTemplateEngine(new FreemarkerTemplateEngine());
        mpg.execute();
    }
}
```