maven 通过执行一些简单命令即可实现上边生命周期的各各过程，比如执行 mvn compile 执行编译、执行 mvn clean 执行清理。

- 一组标准集合

  maven 将整个项目管理过程定义一组标准，比如：通过 maven 构建工程有标准的目录结构，有标准的生命周期阶段、依赖管理有标准的坐标定义等。

- 插件(plugin)目标(goal)

maven 管理项目生命周期过程都是基于插件完成的。

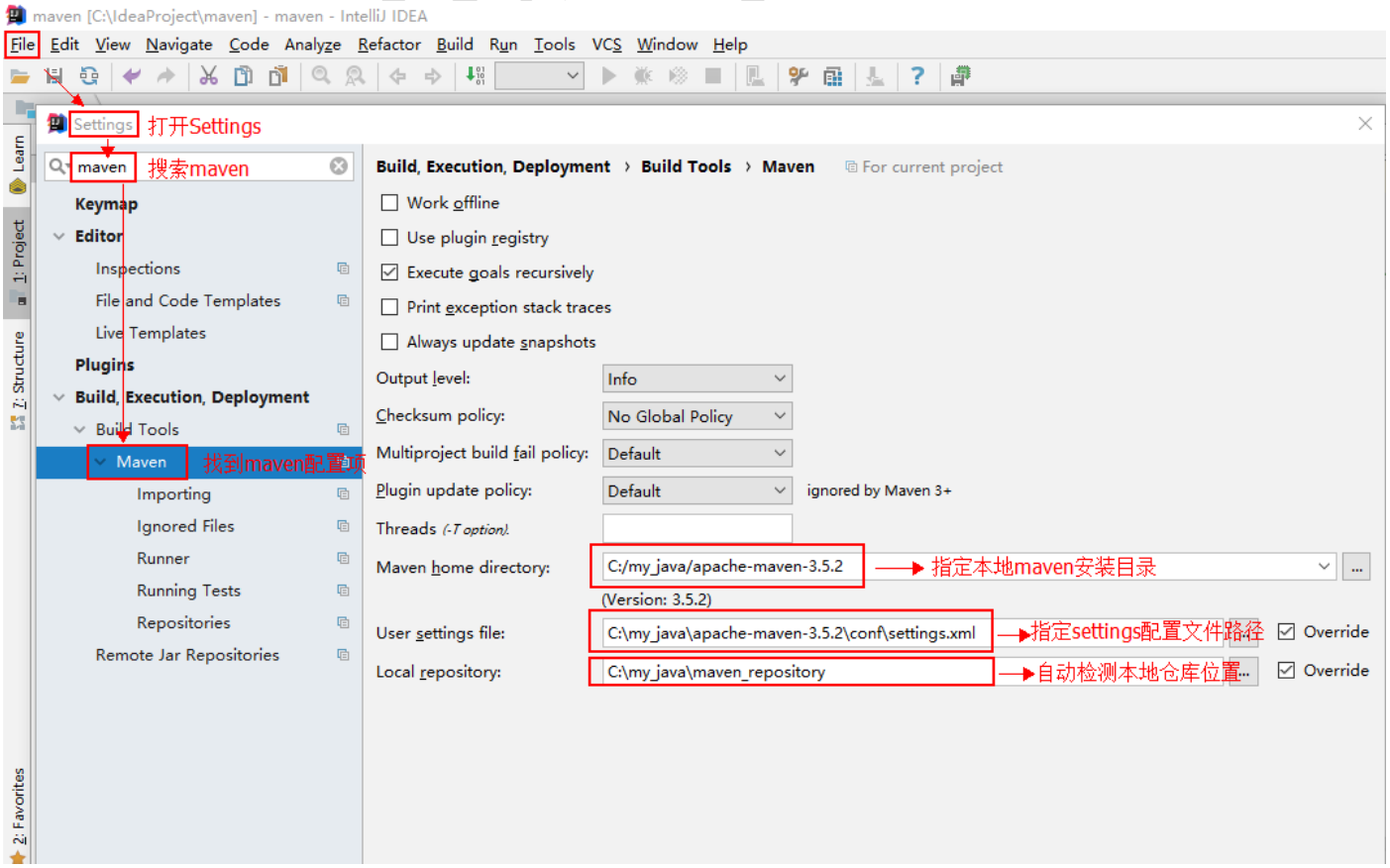
3.2 idea 开发 maven 项目

在实战的环境中，我们都会使用流行的工具来开发项目。

3.2.1 idea 的 maven 配置

3.2.1.1 打开→File→Settings 配置 maven

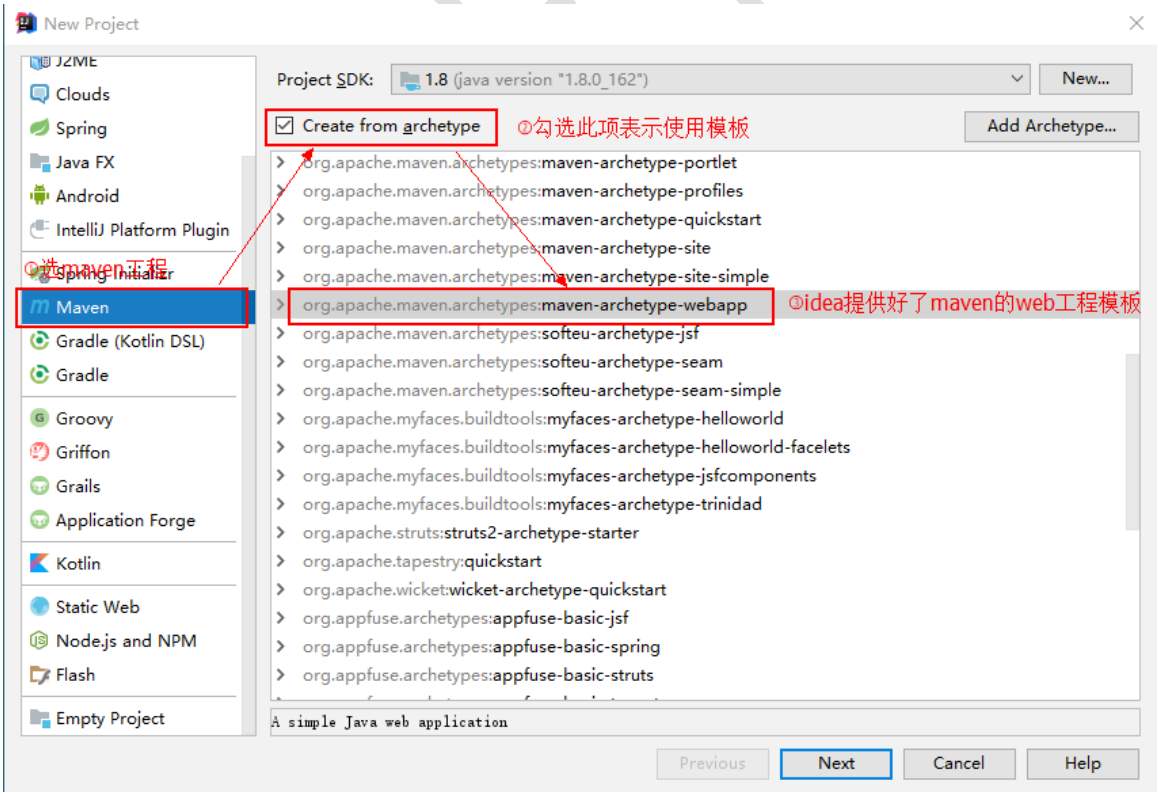
依据图片指示，选择本地 maven 安装目录，指定 maven 安装目录下 conf 文件夹中 settings 配置文件。



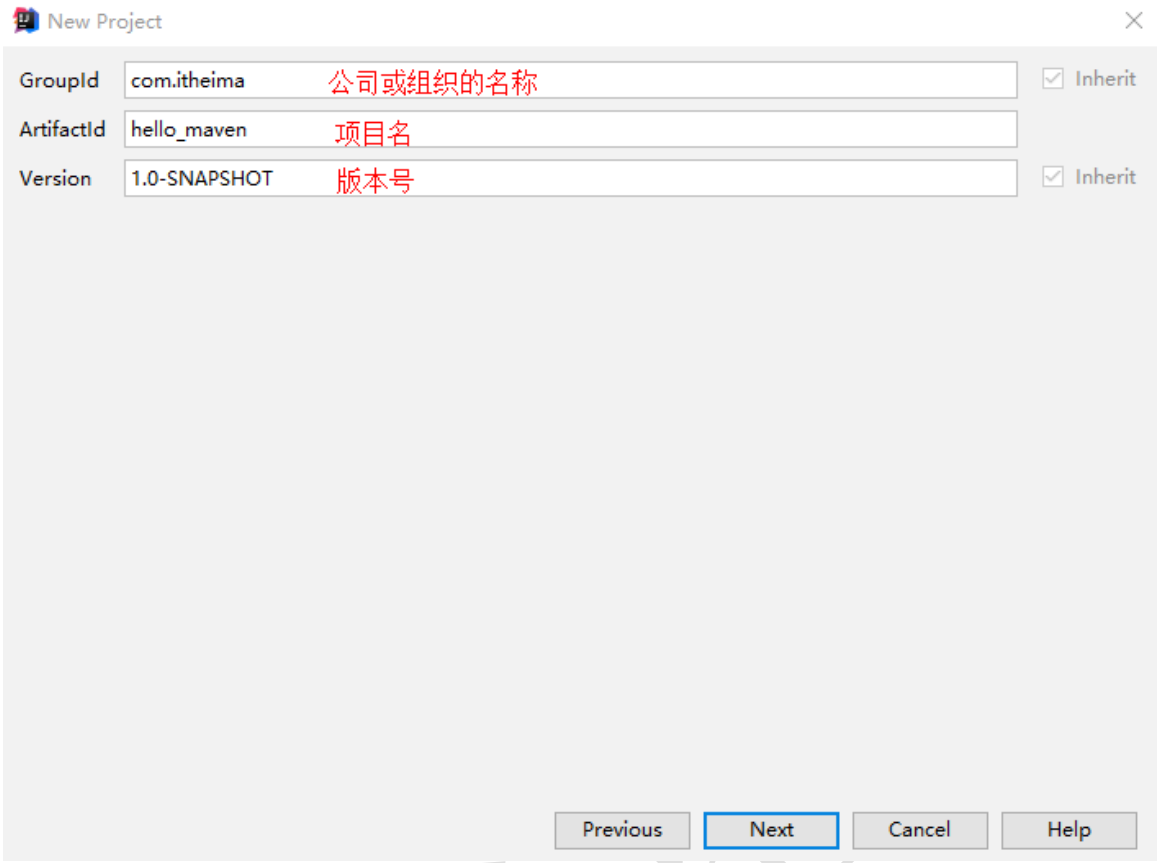
3.2.2 idea 中创建一个 maven 的 web 工程

打开 idea，选择创建一个新工程

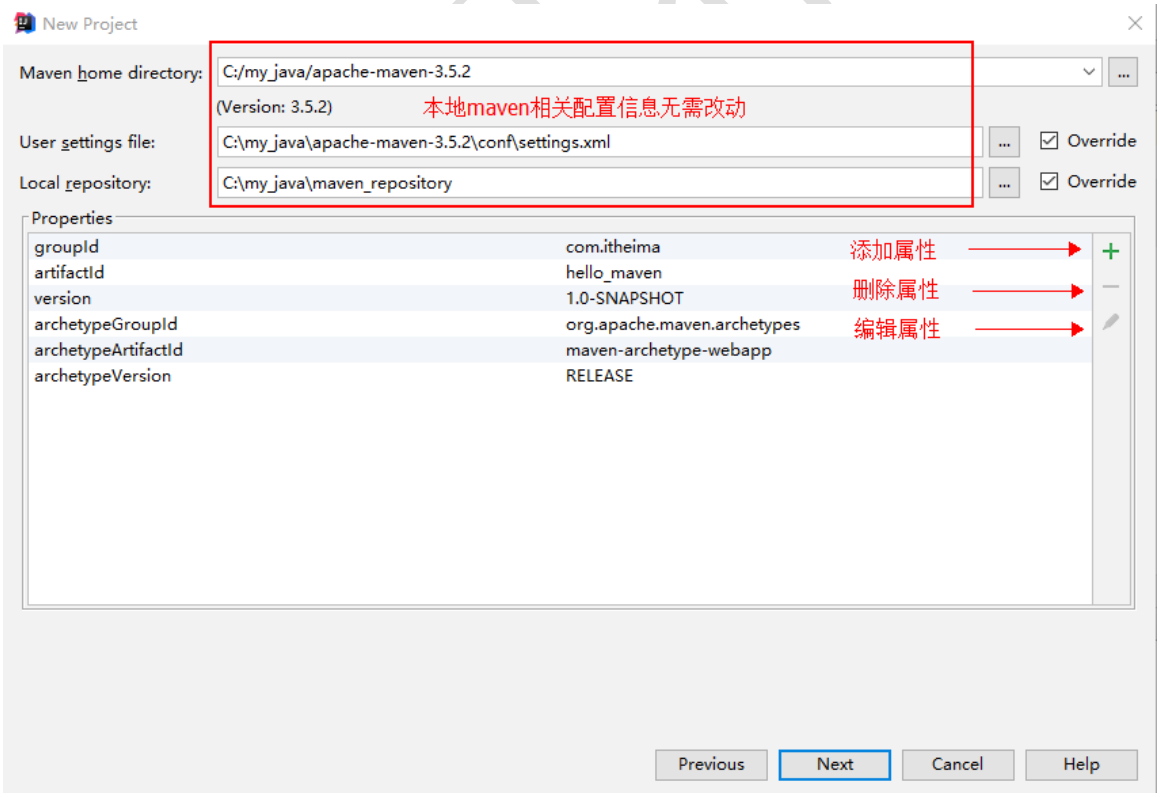
选择 idea 提供好的 maven 的 web 工程模板



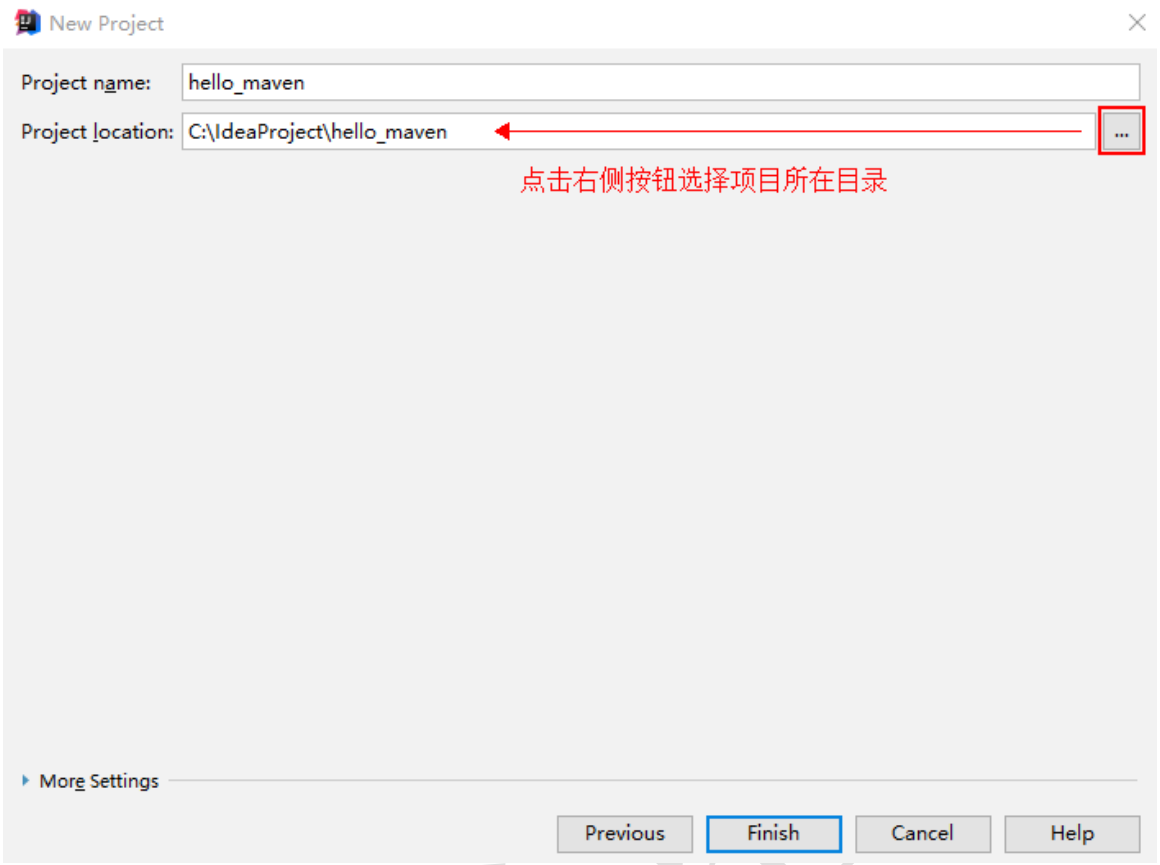
点击 Next 填写项目信息



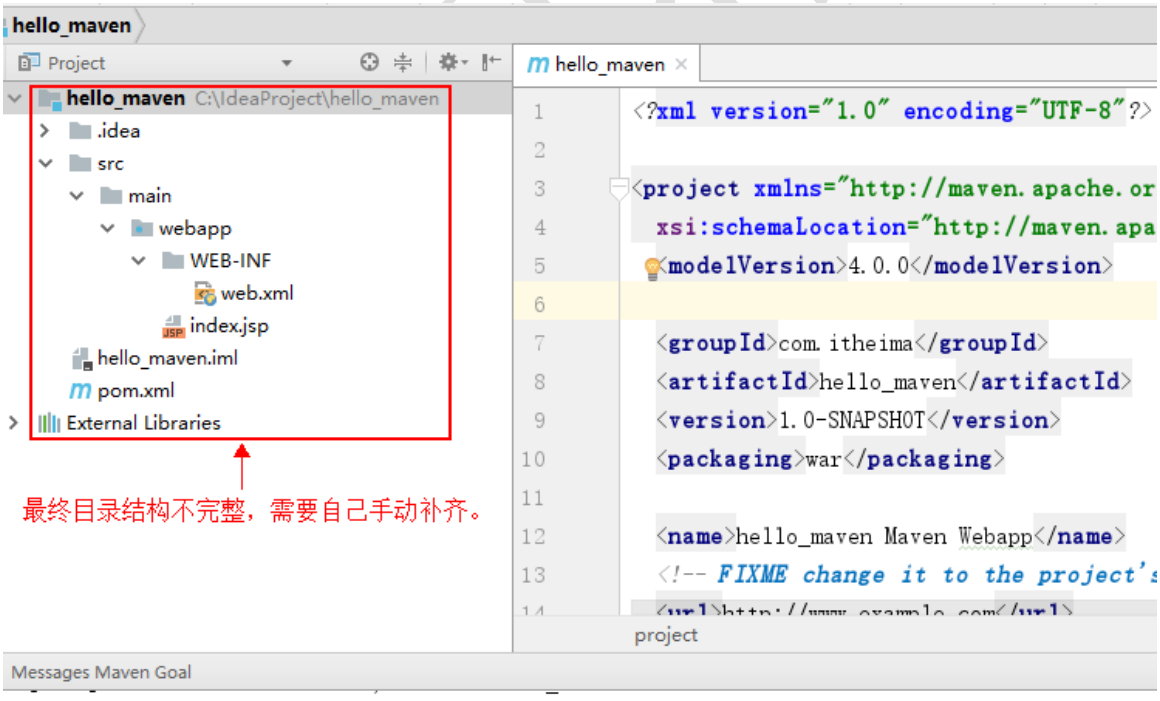
点击 Next，此处不做改动。

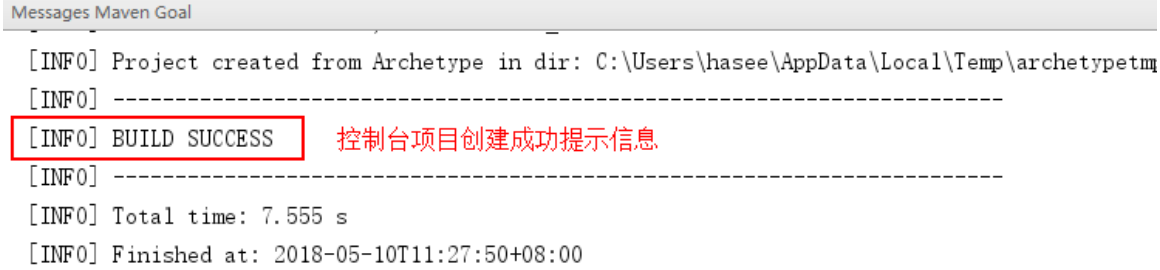


点击 Next 选择项目所在目录

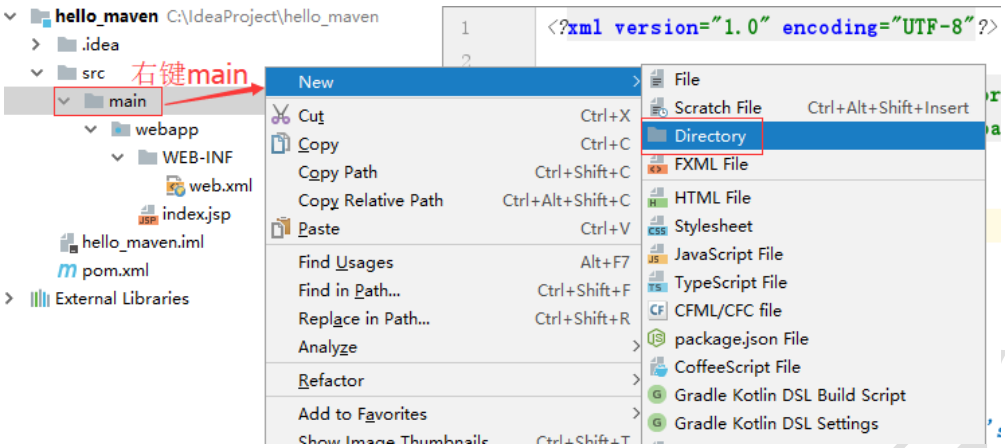


点击 Finish 后开始创建工程，耐心等待，直到出现如下界面。

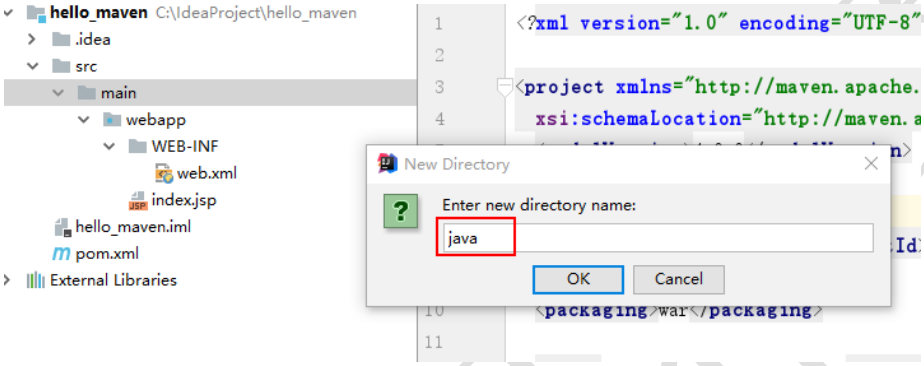




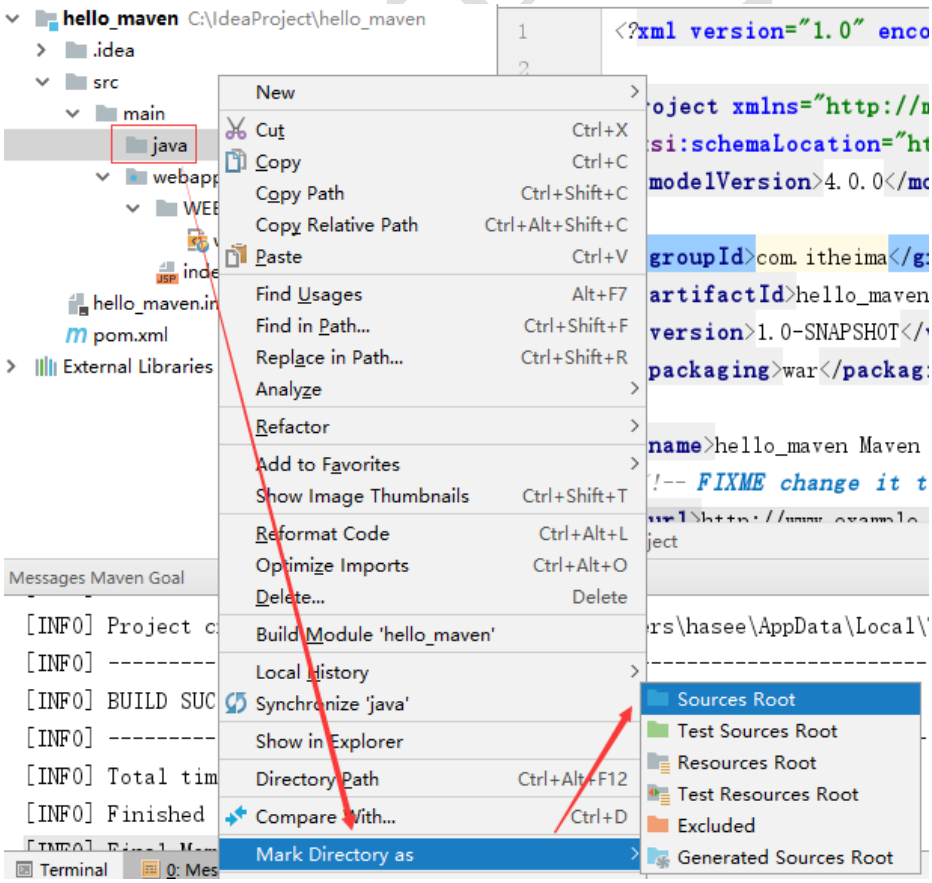
手动添加 src/main/java 目录，如下图右键 main 文件夹→New→Directory



创建一个新的文件夹命名为 java



点击 OK 后，在新的文件夹 java 上右键→Make Directory as→Sources Root



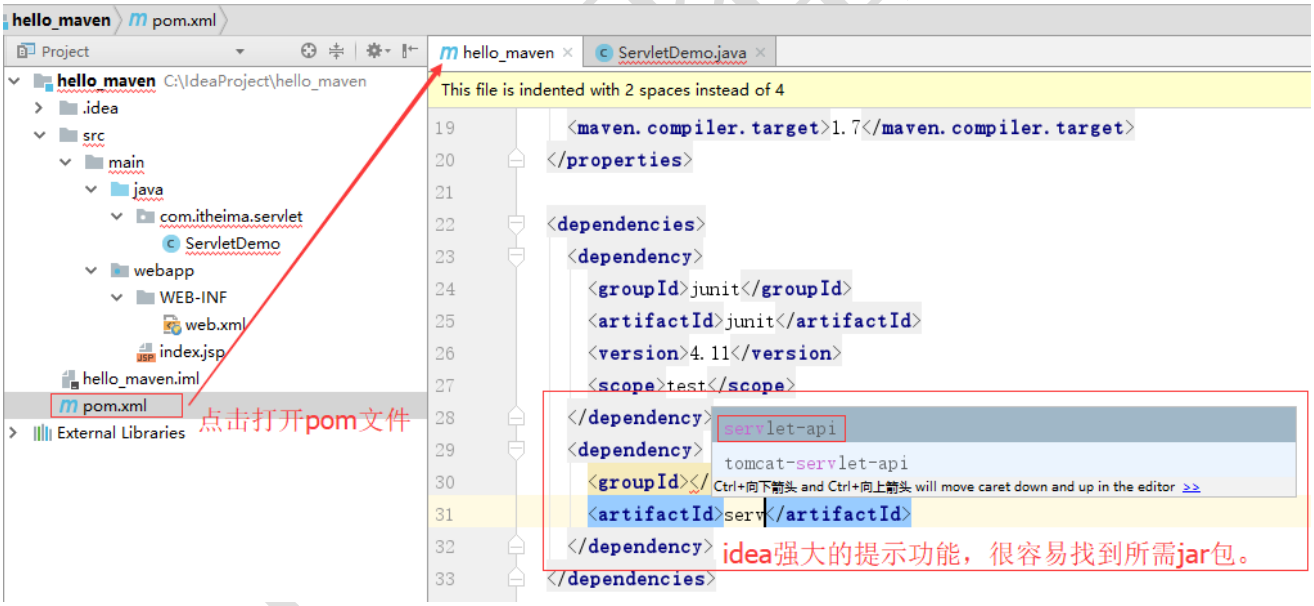
3.2.2.1 创建一个 Servlet

src/java/main 创建了一个 Servlet，但报错

要解决问题，就是要将 servlet-api-xxx.jar 包放进来，作为 maven 工程应当添加 servlet 的坐标，从而导入它的 jar

3.2.2.2 在 pom.xml 文件添加坐标

直接打开 hello_maven 工程的 pom.xml 文件，再添加坐标



添加 jar 包的坐标时，还可以指定这个 jar 包将来的作用范围。

每个 maven 工程都需要定义本工程的坐标，坐标是 maven 对 jar 包的身份定义，比如：入门程序的坐标定义如下：

```
<!--项目名称，定义为组织名+项目名，类似包名-->
<groupId>com.itheima</groupId>
<!-- 模块名称 -->
<artifactId>hello_maven</artifactId>
<!-- 当前项目版本号，snapshot 为快照版本即非正式版本，release 为正式发布版本 -->
```

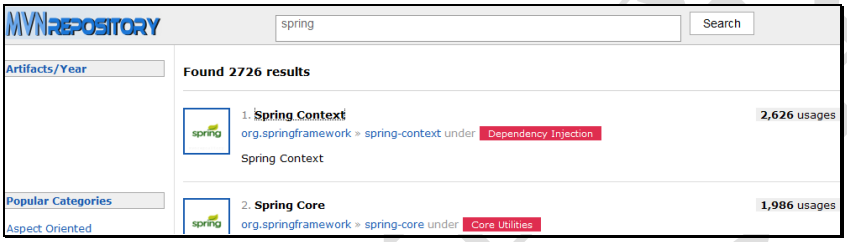
<version>0.0.1-SNAPSHOT</version>
<packaging> ：打包类型
jar：执行 package 会打成 jar 包
war：执行 package 会打成 war 包
pom ：用于 maven 工程的继承，通常父工程设置为 pom

3.2.2.3 坐标的来源方式

添加依赖需要指定依赖 jar 包的坐标，但是很多情况我们是不知道 jar 包的的坐标，可以通过如下方式查询：

3.2.2.3.1 从互联网搜索

http://search.maven.org/
http://mvnrepository.com/
网站搜索示例：



3.2.3 依赖范围

A 依赖 B，需要在 A 的 pom.xml 文件中添加 B 的坐标，添加坐标时需要指定依赖范围，依赖范围包括：

- ✓ compile：编译范围，指 A 在编译时依赖 B，此范围为默认依赖范围。编译范围的依赖会用在编译、测试、运行，由于运行时需要所以编译范围的依赖会被打包。

- ✓ provided：provided 依赖只有在当 JDK 或者一个容器已提供该依赖之后才使用， provided 依赖在编译和测试时需要，在运行时不需要，比如：servlet api 被 tomcat 容器提供。

- ✓ runtime：runtime 依赖在运行和测试系统的时候需要，但在编译的时候不需要。比如：jdbc 的驱动包。由于运行时需要所以 runtime 范围的依赖会被打包。

- ✓ test：test 范围依赖 在编译和运行时都不需要，它们只有在测试编译和测试运行阶段可用，比如：junit。由于运行时不需要所以 test 范围依赖不会被打包。

- ✓ system：system 范围依赖与 provided 类似，但是你必须显式的提供一个对于本地系统中 JAR 文件的路径，需要指定 systemPath 磁盘路径，system 依赖不推荐使用。

依赖范围	对于编译classpath有效	对于测试classpath有效	对于运行时classpath有效	例子
compile	Y	Y	Y	spring-core
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	JDBC驱动
system	Y	Y	-	本地的，Maven仓库之外的类库

在 maven-web 工程中测试各各 scop。

测试总结：

- ✓ 默认引入 的 jar 包 ------- compile 【默认范围 可以不写】（编译、测试、运行 都有效 ）
- ✓ servlet-api 、jsp-api ------- provided （编译、测试 有效， 运行时无效 防止和 tomcat 下 jar 冲突）
- ✓ jdbc 驱动 jar 包 ---- runtime （测试、运行 有效 ）
- ✓ junit ----- test （测试有效）

依赖范围由强到弱的顺序是：compile>provided>runtime>test

3.2.4 项目中添加的坐标

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

3.2.5 设置 jdk 编译版本

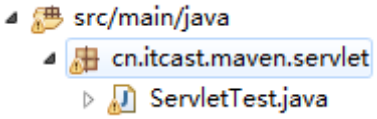
本教程使用 jdk1.8，需要设置编译版本为 1.8，这里需要使用 maven 的插件来设置：
在 pom.xml 中加入：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
```

```
            </plugins>
        </build>
```

3.2.6 编写 servlet

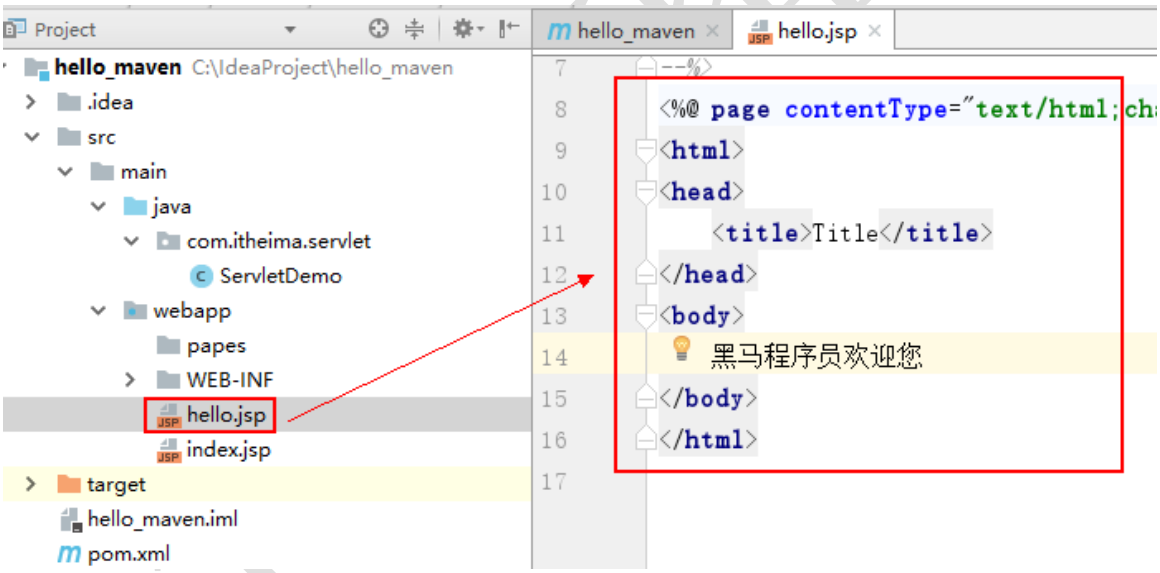
在 src/main/java 中创建 ServletTest



内容如下

```
public class ServletDemo extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
        req.getRequestDispatcher( s: "/hello.jsp").forward(req, resp);
    }
}
```

3.2.7 编写 jsp



3.2.8 在 web.xml 中配置 servlet 访问路径

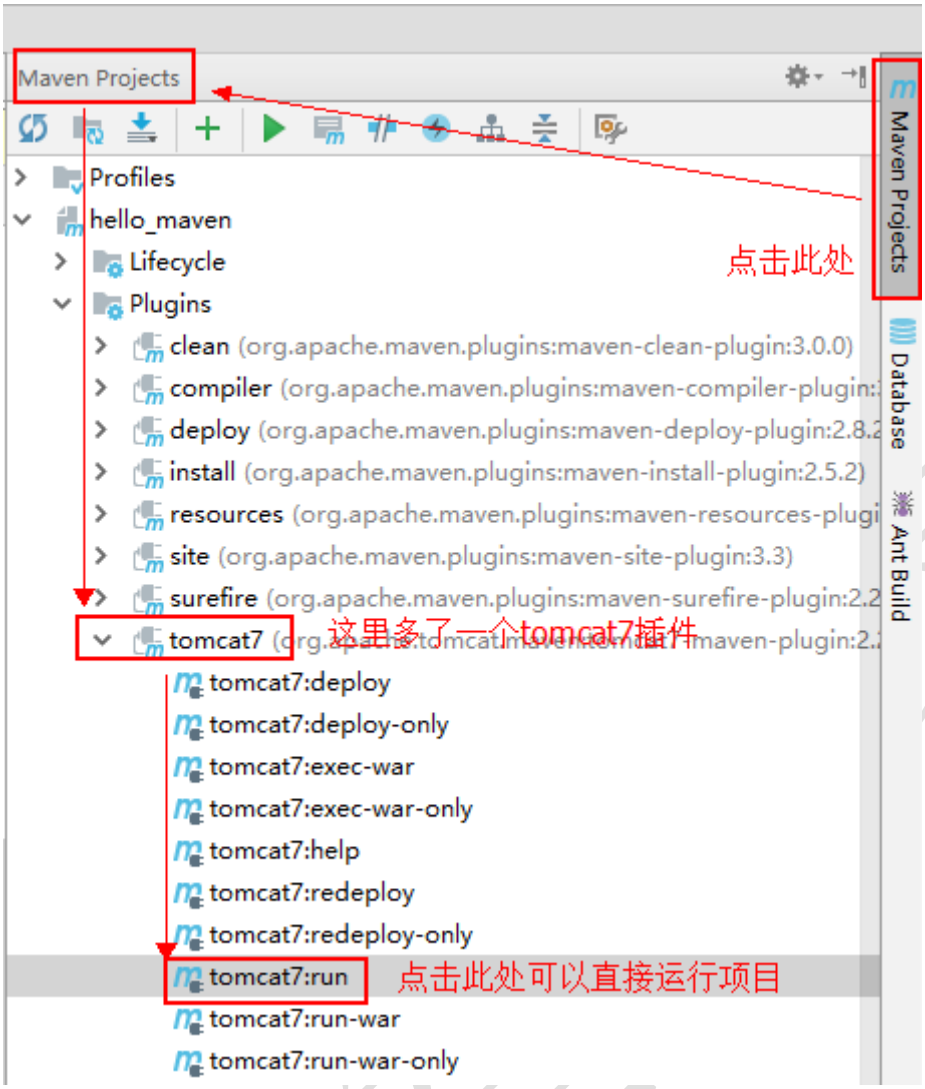
```
<servlet>
  <servlet-name>servletDemo</servlet-name>
  <servlet-class>com.itheima.servlet.ServletDemo</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>servletDemo</servlet-name>
  <url-pattern>/maven</url-pattern>
</servlet-mapping>
```

3.2.9 添加 tomcat7 插件

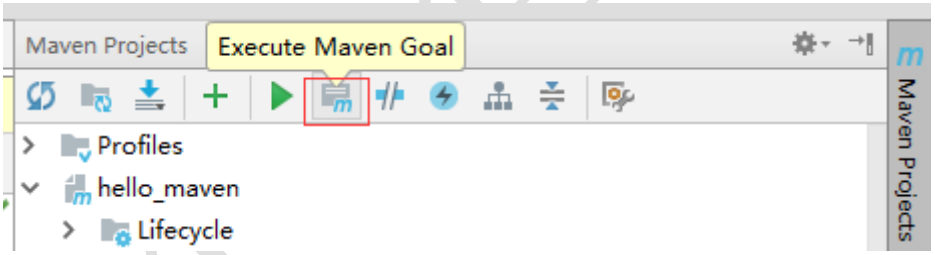
在 pom 文件中添加如下内容

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <port>8080</port>   端口号
    <path>/</path>      访问路径
  </configuration>
</plugin>
```

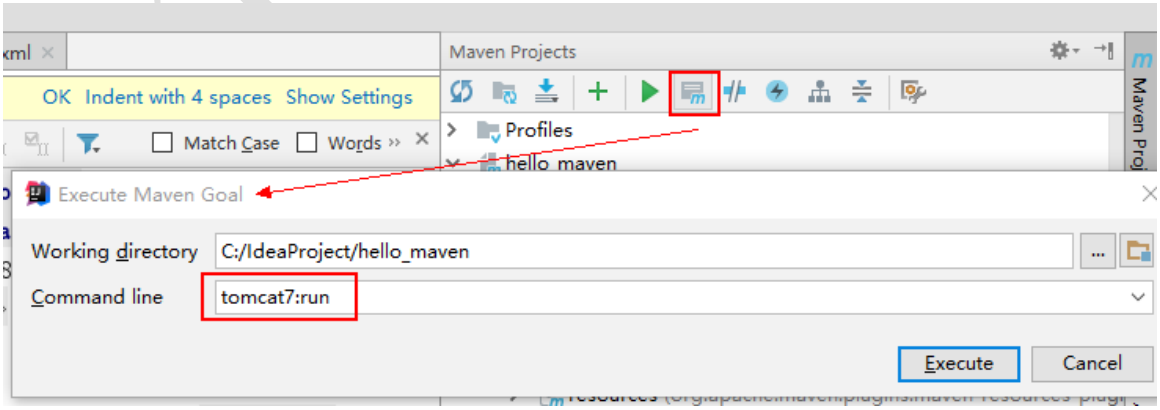
此时点击 idea 最右侧 Maven Projects，
就可以看到我们新添加的 tomcat7 插件
双击 tomcat7 插件下 tomcat7:run 命令直接运行项目



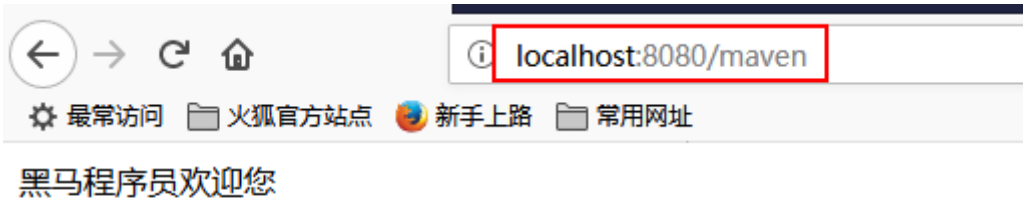
也可以直接点击如图按钮，手动输入 tomc7:run 命令运行项目



点击后弹出如下图窗口



3.2.10运行结果



第4章 maven 工程运行调试

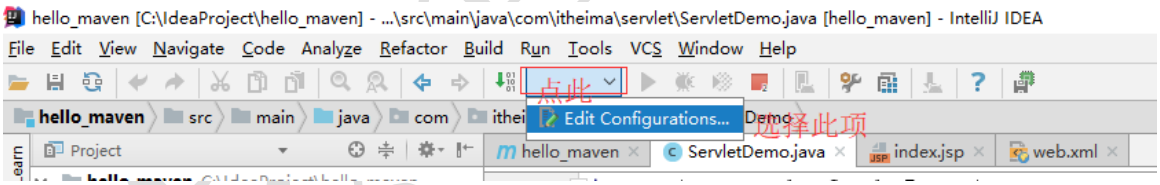
4.1 端口占用处理

重新执行 tomcat:run 命令重启工程，重启之前需手动停止 tomcat，否则报下边的错误：

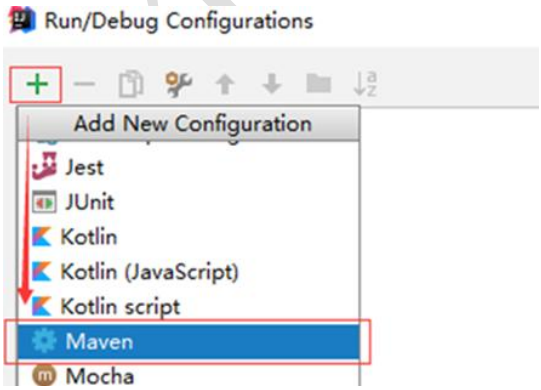
```
严重: Failed to initialize end point associated with ProtocolHandler ["http-bio-8080"]
java.net.BindException: Address already in use: JVM_Bind <null>:8080
    at org.apache.tomcat.util.net.JIoEndpoint.bind(JIoEndpoint.java:407)
```

4.2 断点调试

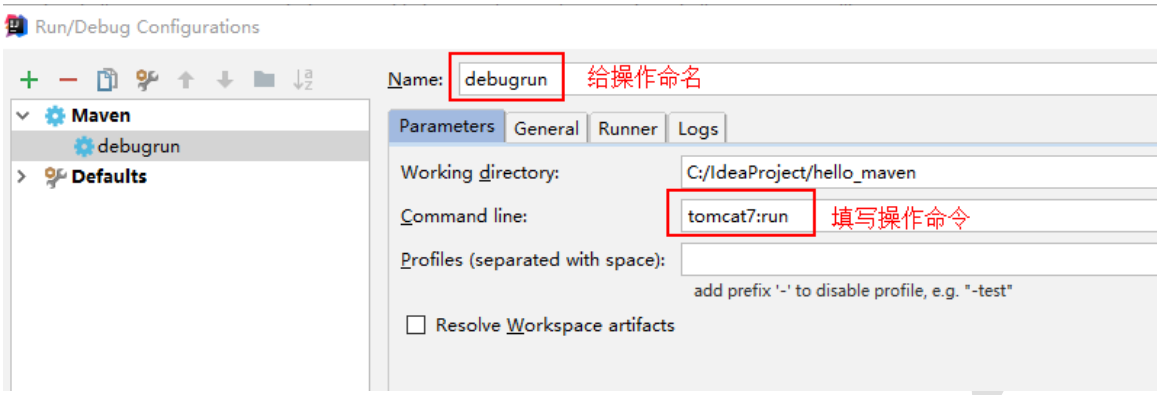
点击如图所示选项



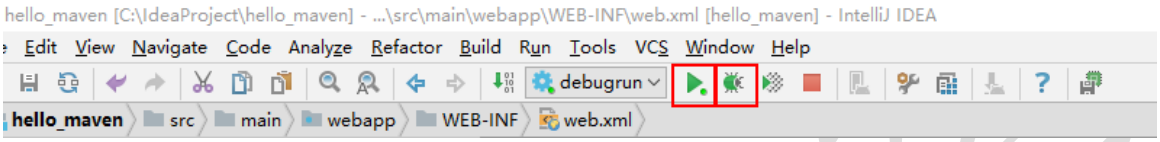
在弹出框中点击如图加号按钮找到 maven 选项



在弹出窗口中填写如下信息



完成后先 Apply 再 OK 结束配置后，可以在主界面找到我们刚才配置的操作名称。



如上图红框选中的两个按钮，左侧是正常启动，右侧是 debug 启动。

第5章 总结

5.1 maven 仓库

1、maven 仓库的类型有哪些？
2、maven 工程查找仓库的流程是什么？
3、本地仓库如何配置？

5.2 常用的 maven 命令

常用 的 maven 命令包括：
compile：编译
clean：清理
test：测试
package：打包
install：安装

5.3 坐标定义

在 pom.xml 中定义坐标，内容包括：groupId、artifactId、version，详细内容如下：

```
<!--项目名称，定义为组织名+项目名，类似包名-->
<groupId>cn.itcast.maven</groupId>
<!-- 模块名称 -->
```

<artifactId>maven-first</artifactId>
<!-- 当前项目版本号，snapshot 为快照版本即非正式版本，release 为正式发布版本 -->
<version>0.0.1-SNAPSHOT</version>
<packaging > ： 打包类型
jar：执行 package 会打成 jar 包
war：执行 package 会打成 war 包
pom ： 用于 maven 工程的继承，通常父工程设置为 pom

5.4 pom 基本配置

pom.xml 是 Maven 项目的核心配置文件，位于每个工程的根目录，基本配置如下：

<project > ： 文件的根节点 .
<modelversion > ： pom.xml 使用的对象模型版本
<groupId > ： 项目名称，一般写项目的域名
<artifactId > ： 模块名称，子项目名或模块名称
<version > ： 产品的版本号 .
<packaging > ： 打包类型，一般有 jar、war、pom 等
<name > ： 项目的显示名，常用于 Maven 生成的文档。
<description > ： 项目描述，常用于 Maven 生成的文档
<dependencies> ： 项目依赖构件配置，配置项目依赖构件的坐标
<build> ： 项目构建配置，配置编译、运行插件等。