

贝密游戏服务端管理系统

(BeiMi)

安 装 手 册

贝密游戏

二〇一七年九月

目录

目录.....	2
1. Maven 安装及配置	3
1.1 Maven 安装	3
1.2 Maven 基础配置	3
1.2.1 使用 Maven 编译运行贝密游戏项目	4
1.2.2 pom.xml 详细配置	5
1.2.3 安装依赖的 JAR 包	5
1.2.4 使用 Maven 编译 BeiMi 项目	7
1.2.5 运行贝密游戏项目	7
1.2.6 运行 BeiMi 项目	9
1.3 MyEclipse2014 安装配置	10
1.3.1 MyEclipse2014 安装	10
1.3.2 MyEclipse2014 中 Maven 配置	11
1.4 BeiMi 项目工程 Maven 化	17
1.4.1 Maven 标准目录结构说明	17
1.4.2 pom.xml 详细配置	18
1.4.3 在 MyEclipse 中导入工程	18
1.4.4 运行 BeiMi 项目	21
2. 选读内容 (maven 使用配置及命令)	25
3.1 pom.xml 配置说明	25

1.Maven 安装及配置

1.1 Maven 安装

a) 解压 apache-maven-3.3.1-bin.zip 至某目录，比如 F:\Program

Files\apache-maven-3.3.1；

说明：apache-maven-3.3.1 支持jdk1.7+。

b) 配置系统环境变量 M2_HOME，值为 F:\Program

Files\apache-maven-3.3.1；

c) 在系统环境变量 path 中尾部添加%M2_HOME%\bin;

1.2 Maven 基础配置

配置中央仓库(远程仓库)的镜像，作用是为了稳定、快速的访问中央仓库（可以使用阿里云提供的 Maven 仓库）。

```
<settings>

.....

<mirrors>

  <mirror>

    <id>nexus-aliyun</id>

    <mirrorOf>*</mirrorOf>

    <name>Nexus aliyun</name>
```

```
<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

</mirrors>

<profiles>

  <profile>

    .....

  </profile>

</profiles>

.....

<settings>
```

1.2.1 使用 Maven 编译运行贝密游戏项目

从码云上下载 BeiMi 代码工程，下载地址为：

<https://gitee.com/beimigame/beimi>

BeiMi 源码工程目录结构如下

目录	说明
src	源码目录
src/main	主代码
src/main/java	主代码 java 文件
src/main/resources	主代码资源文件
src/test	测试代码

src/test/resources	测试代码资源文件
target	构建输出目录
pom.xml	Maven 工程文件
Client	客户端工程
Ui	贝密游戏的客户端美术资源（含 PSD 源文件）
Version	贝密游戏客户端工程（Cocos Creator）

1.2.2 pom.xml 详细配置

BeiMi 默认配置的 pom.xml 文件无需做修改。

1.2.3 安装依赖的 JAR 包

进入 BeiMi 项目根目录下，如下图所示：

```

D:\wordspace\program\beimi\chess>dir
驱动器 D 中的卷是 新加卷
卷的序列号是 806A-B2D0

D:\wordspace\program\beimi\chess 的目录
2017/08/26  23:09    <DIR>        .
2017/08/26  23:09    <DIR>        ..
2017/08/29  20:13             1,133 .classpath
2017/08/26  23:09    <DIR>        .myeclipse
2017/08/17  08:44             1,572 .project
2017/08/16  22:58    <DIR>        .settings
2017/08/16  22:58             66 .tern-project
2017/08/13  23:08    <DIR>        client
2017/08/27  11:24    <DIR>        data
2017/09/05  23:10    <DIR>        doc
2017/09/05  22:29             9,006 gc.log
2017/08/13  23:08             10,447 LICENSE
2017/08/03  21:23             10,931 pom.xml
2017/08/13  23:08             2,513 README.md
2017/08/13  23:08    <DIR>        script
2017/08/13  23:09    <DIR>        src
2017/08/16  22:50    <DIR>        target
                7 个文件          35,728 字节
                10 个目录 11,337,879,552 可用字节

D:\wordspace\program\beimi\chess>
  
```

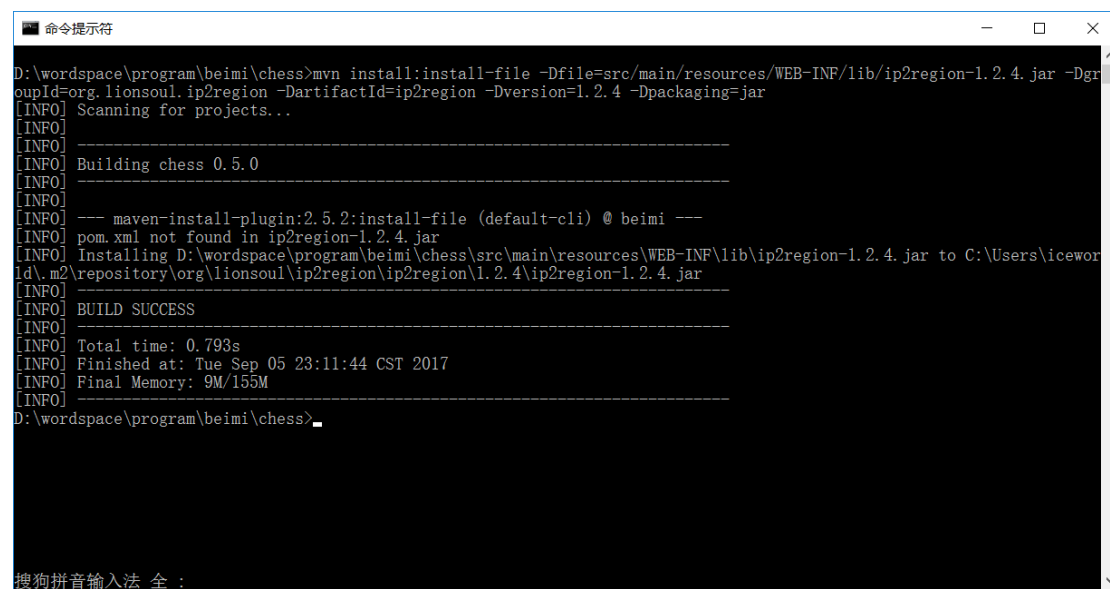
BeiMi 需要使用两个不在 Maven 中央仓库中的第三方 JAR 文件，JAR 文件存放在 BeiMi\src\main\resources\WEB-INF\lib 目录下，依次执行如下命令（执行之前，请确保 Maven 在 PATH 路径中配置正确，并且当前光标位置在 pom.xml

文件同一目录)：

注意：如果本地安装的 Maven 从未下载过项目中需要使用到的 JAR 文件，则可能需要花费较长时间下载 JAR 资源文件，过程中会有详细提示，如果使用的是 Maven 中央仓库，执行下载过程耗时非常长；如果按照前面的说明中修改为使用阿里云的 Maven 仓库，则下载时间相对会缩短。

```
mvn install:install-file  
  
-Dfile=src/main/resources/WEB-INF/lib/ip2region-1.2.4.jar  
  
-DgroupId=org.lionsoul.ip2region -DartifactId=ip2region -Dversion=1.2.4  
  
-Dpackaging=jar
```

执行后提示如下信息：



```
命令提示符  
D:\wordspace\program\beimi\chess>mvn install:install-file -Dfile=src/main/resources/WEB-INF/lib/ip2region-1.2.4.jar -DgroupId=org.lionsoul.ip2region -DartifactId=ip2region -Dversion=1.2.4 -Dpackaging=jar  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----  
[INFO] Building chess 0.5.0  
[INFO] -----  
[INFO] --- maven-install-plugin:2.5.2:install-file (default-cli) @ beimi ---  
[INFO] pom.xml not found in ip2region-1.2.4.jar  
[INFO] Installing D:\wordspace\program\beimi\chess\src\main\resources\WEB-INF\lib\ip2region-1.2.4.jar to C:\Users\icewor  
ld\.m2\repository\org\lionsoul\ip2region\ip2region\1.2.4\ip2region-1.2.4.jar  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 0.793s  
[INFO] Finished at: Tue Sep 05 23:11:44 CST 2017  
[INFO] Final Memory: 9M/155M  
[INFO] -----  
D:\wordspace\program\beimi\chess>
```

```
mvn install:install-file -Dfile=src/main/resources/WEB-INF/lib/jave-1.0.2.jar  
  
-DgroupId=lt.java -DartifactId=jave -Dversion=1.0.2 -Dpackaging=jar
```

执行后提示如下信息：

```
命令提示符
D:\wordspace\program\beimi\chess>mvn install:install-file -Dfile=src/main/resources/WEB-INF/lib/jave-1.0.2.jar -DgroupId=
=lt.jave -DartifactId=jave -Dversion=1.0.2 -Dpackaging=jar
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building chess 0.5.0
[INFO] -----
[INFO] --- maven-install-plugin:2.5.2:install-file (default-cli) @ beimi ---
[INFO] pom.xml not found in jave-1.0.2.jar
[INFO] Installing D:\wordspace\program\beimi\chess\src\main\resources\WEB-INF\lib\jave-1.0.2.jar to C:\Users\iceworld\.m
2\repository\lt\jave\jave\1.0.2\jave-1.0.2.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 0.896s
[INFO] Finished at: Tue Sep 05 23:12:02 CST 2017
[INFO] Final Memory: 9M/155M
[INFO] -----
D:\wordspace\program\beimi\chess>
```

1.2.4 使用 Maven 编译 BeiMi 项目

在 BeiMi 项目根目录下执行编译命令：

```
mvn compile
```

```
命令提示符
D:\wordspace\program\beimi\chess>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building chess 0.5.0
[INFO] -----
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ beimi ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 991 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ beimi ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 2.955s
[INFO] Finished at: Tue Sep 05 23:13:34 CST 2017
[INFO] Final Memory: 28M/359M
[INFO] -----
D:\wordspace\program\beimi\chess>
```

提示成功后即可执行运行命令。

1.2.5 运行贝密游戏项目

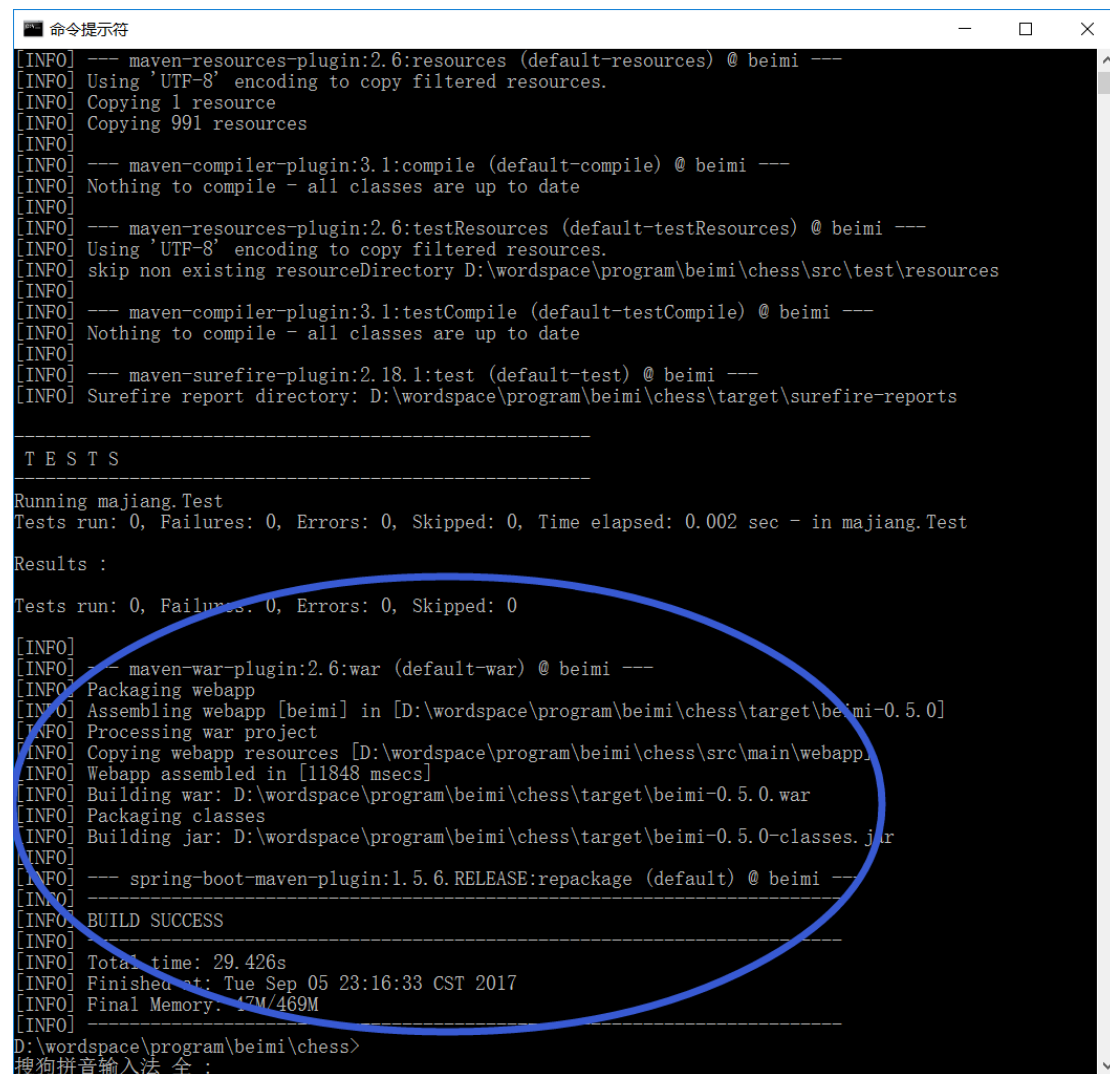
运行贝密游戏服务端有两种方式，第一种是使用 Maven 命令运行文件，

```
mvn spring-boot:start
```

第二种方式是使用 Maven 打包命令生成最终运行包文件，命令如下：

```
mvn package
```

如下图所示：

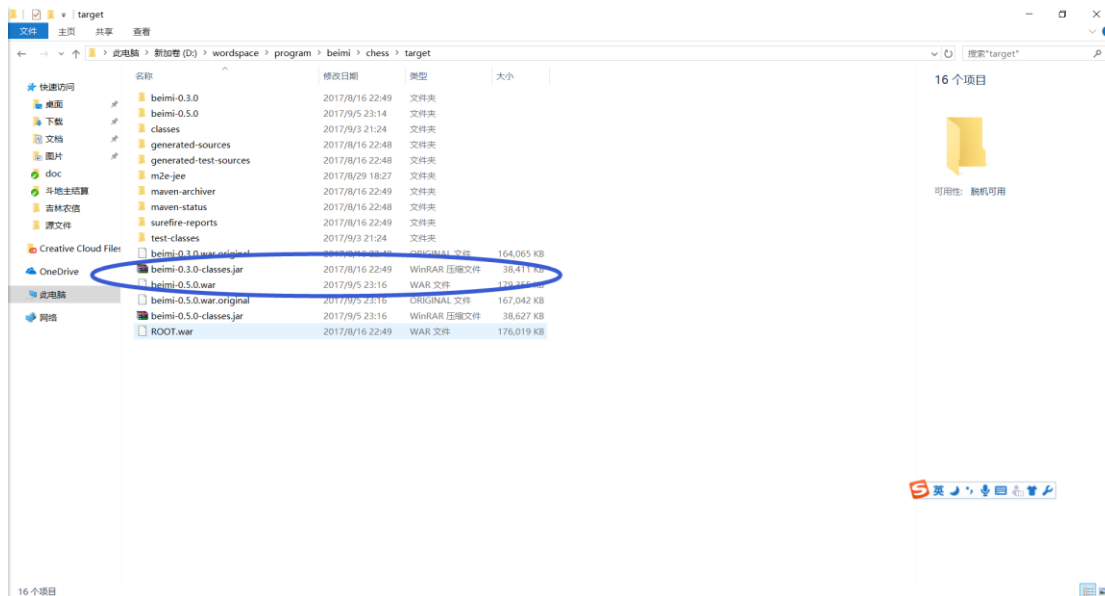


```
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ beimi ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 991 resources
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ beimi ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ beimi ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\wordspace\program\beimi\chess\src\test\resources
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ beimi ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ beimi ---
[INFO] Surefire report directory: D:\wordspace\program\beimi\chess\target\surefire-reports

-----
T E S T S
-----
Running majiang.Test
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec - in majiang.Test
Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- maven-war-plugin:2.6:war (default-war) @ beimi ---
[INFO] Packaging webapp
[INFO] Assembling webapp [beimi] in [D:\wordspace\program\beimi\chess\target\beimi-0.5.0]
[INFO] Processing war project
[INFO] Copying webapp resources [D:\wordspace\program\beimi\chess\src\main\webapp]
[INFO] Webapp assembled in [11848 msecs]
[INFO] Building war: D:\wordspace\program\beimi\chess\target\beimi-0.5.0.war
[INFO] Packaging classes
[INFO] Building jar: D:\wordspace\program\beimi\chess\target\beimi-0.5.0-classes.jar
[INFO] --- spring-boot-maven-plugin:1.5.6.RELEASE:repackage (default) @ beimi ---
[INFO] BUILD SUCCESS
[INFO] Total time: 29.426s
[INFO] Finished at: Tue Sep 05 23:16:33 CST 2017
[INFO] Final Memory: 47M/469M
[INFO] D:\wordspace\program\beimi\chess>
搜狗拼音输入法 全：
```

查看 target 目录下打包完成的 jar 运行文件



完成！

1.2.6 运行 BeiMi 项目

将 beimi-0.5.0.war 文件拷贝到一个空的目录下，执行如下命令：

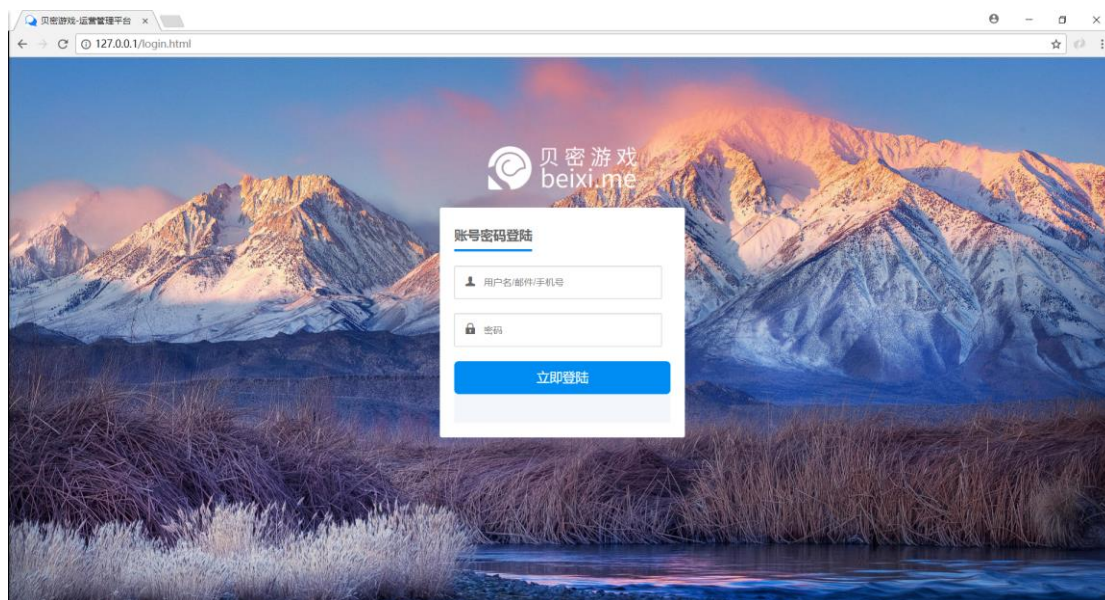
```
java -Xms1240m -Xmx1240m -Xmn450m -XX:PermSize=512M  
-XX:MaxPermSize=512m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC  
-XX:+UseTLAB -XX:NewSize=128m -XX:MaxNewSize=128m  
-XX:MaxTenuringThreshold=0 -XX:SurvivorRatio=1024  
-XX:+UseCMSInitiatingOccupancyOnly  
-XX:CMSInitiatingOccupancyFraction=60 -Djava.awt.headless=true  
-XX:+PrintGCDetails -Xloggc:gc.log -XX:+PrintGCtimeStamps -jar  
beimi-0.5.0.war
```

执行后出现如下信息：

```
命令提示符 - java -Xms1240m -Xmx1240m -Xmn540m -XX:PermSize=512M -XX:MaxPermSize=512M -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+UseTLAB -XX:NewSize=128m -XX:MaxNewSize=128m -XX:MaxTenuringThreshold=10
2017-09-05 23:19:46.938 [main] INFO o.s.b.a.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()
2017-09-05 23:19:47.229 [main] INFO o.s.b.a.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()
2017-09-05 23:19:46.939 [main] INFO o.s.b.a.actuate.endpoint.mvc.EndpointHandlerMapping - Mapped "[/metrics/{name}/{id}]" to public java.lang.Object org.springframework.boot.actuate.endpoint.mvc.EndpointMvcAdapter.invoke()
2017-09-05 23:19:47.235 [main] INFO o.s.w.servlet.view.freemarker.FreeMarkerConfigurer - ClassTemplateLoader for Spring macros added to FreeMarker configuration
2017-09-05 23:19:47.512 [main] INFO o.s.jax.export.annotation.AnnotationMBeanExporter - Registering beans for JMX exposure on startup
2017-09-05 23:19:47.520 [main] INFO o.s.jax.export.annotation.AnnotationMBeanExporter - Bean with name 'dataSource' has been autodetected for JMX exposure
2017-09-05 23:19:47.539 [main] INFO o.s.jax.export.annotation.AnnotationMBeanExporter - Located MBean 'dataSource': registering with JMX server as MBean [com.alibaba.druid.pool:name=DataSource,type=DruidDataSource]
2017-09-05 23:19:47.543 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Registering beans for JMX exposure on startup
2017-09-05 23:19:47.557 [main] INFO o.s.context.support.DefaultLifecycleProcessor - Starting beans in phase 0
2017-09-05 23:19:47.565 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'auditEventsEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=auditEventsEndpoint]
2017-09-05 23:19:47.615 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'requestMappingEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=requestMappingEndpoint]
2017-09-05 23:19:47.654 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'environmentEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=environmentEndpoint]
2017-09-05 23:19:47.665 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'healthEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=healthEndpoint]
2017-09-05 23:19:47.673 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'beansEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=beansEndpoint]
2017-09-05 23:19:47.684 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'infoEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=infoEndpoint]
2017-09-05 23:19:47.689 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'loggersEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=loggersEndpoint]
2017-09-05 23:19:47.702 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'metricsEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=metricsEndpoint]
2017-09-05 23:19:47.708 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'traceEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=traceEndpoint]
2017-09-05 23:19:47.717 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'dumpEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=dumpEndpoint]
2017-09-05 23:19:47.726 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'autoConfigurationReportEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=autoConfigurationReportEndpoint]
2017-09-05 23:19:48.113 [main] INFO o.s.b.a.actuate.endpoint.jmx.EndpointMBeanExporter - Located managed bean 'configurationPropertiesReportEndpoint': registering with JMX server as MBean [org.springframework.boot.actuate.endpoint.name=configurationPropertiesReportEndpoint]
2017-09-05 23:19:48.319 [main] INFO o.s.h.hql.internal.QueryTranslatorFactoryInitiator - HHH0000397: Using ASTQueryTranslatorFactory
2017-09-05 23:19:49.081 [main] INFO org.cache2k.core.Cache2kCoreProviderImpl - cache2k starting, version=1.0.1.Final, build=undefined, defaultImplementation=HeapCache
2017-09-05 23:19:49.230 [main] INFO c.h.internal.partition.impl.PartitionStateManager - [392:163:1:153]:[BeIMI] (3-8) Initializing cluster partition table arrangement...
2017-09-05 23:19:50.595 [main] INFO org.apache.coyote.http11.Http11NioProtocol - Initializing ProtocolHandler ["http-nio-80"]
2017-09-05 23:19:50.595 [main] INFO org.apache.coyote.http11.Http11NioProtocol - Starting ProtocolHandler ["http-nio-80"]
2017-09-05 23:19:50.604 [main] INFO org.apache.tomcat.util.net.NioSelectorPool - Using a shared selector for servlet write/read
2017-09-05 23:19:50.654 [main] INFO o.s.b.e.c.tomcat.TomcatEmbeddedServletContainer - Tomcat started on port(s): 80 (http)
2017-09-05 23:19:50.667 [main] INFO com.corundumstudio.socketio.SocketIOServer - Session store / pubsub factory used: MemoryStoreFactory (local session store only)
2017-09-05 23:19:52.750 [nioEventLoopGroup-2-1] INFO com.corundumstudio.socketio.SocketIOServer - SocketIO server started at port: 9081
2017-09-05 23:19:52.757 [main] INFO com.beimi.Application - Started Application in 40.058 seconds (JVM running for 40.946)
```

系统正常启动，启动成功！BeiMi 默认的访问端口是 80

访问浏览器地址：<http://127.0.0.1:80>，出现如下界面即访问成功：



1.3 MyEclipse2014 安装配置

1.3.1 MyEclipse2014 安装

- 从服务器上将 myeclipse-pro-2014-GA-offline-installer-windows.exe 下载到本地，双击该文件将 myeclipse 安装至某目录。

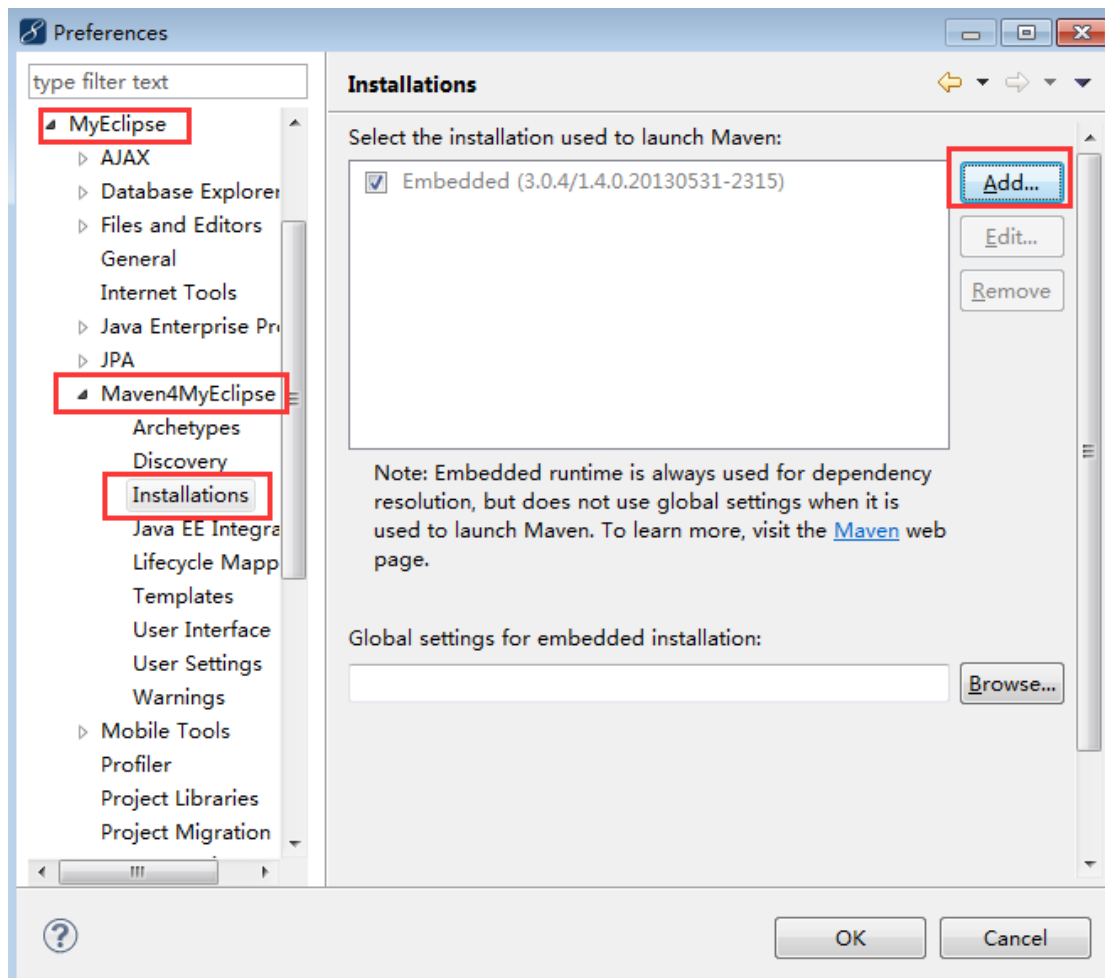
b) 找到安装目录，双击 myeclipse.exe，第一次启动 myeclipse，需要注册码。Myeclipse2014 激活教程.rar，参照教程进行破解。

c) 重新启动 MyEclipse2014。

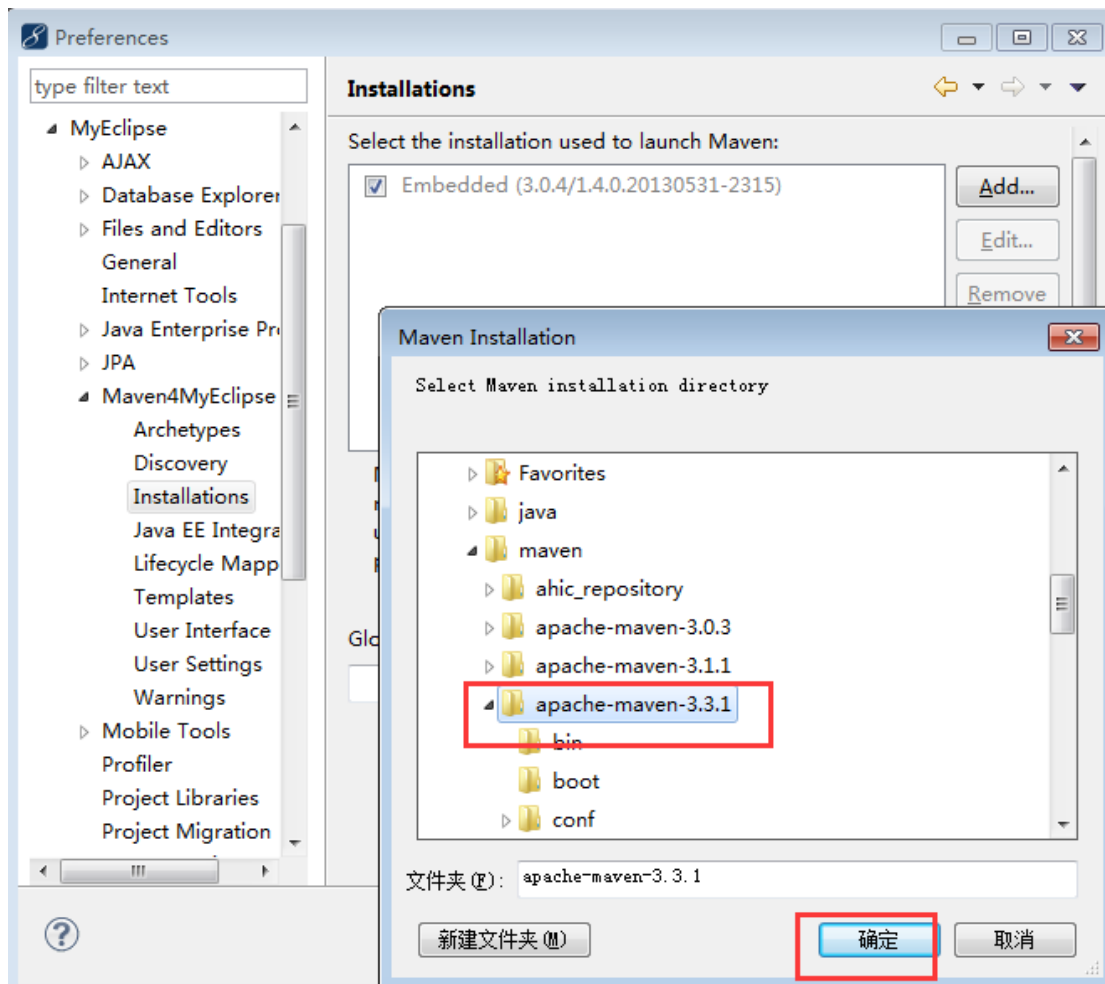
1.3.2 MyEclipse2014 中 Maven 配置

1 设置 Maven 目录：

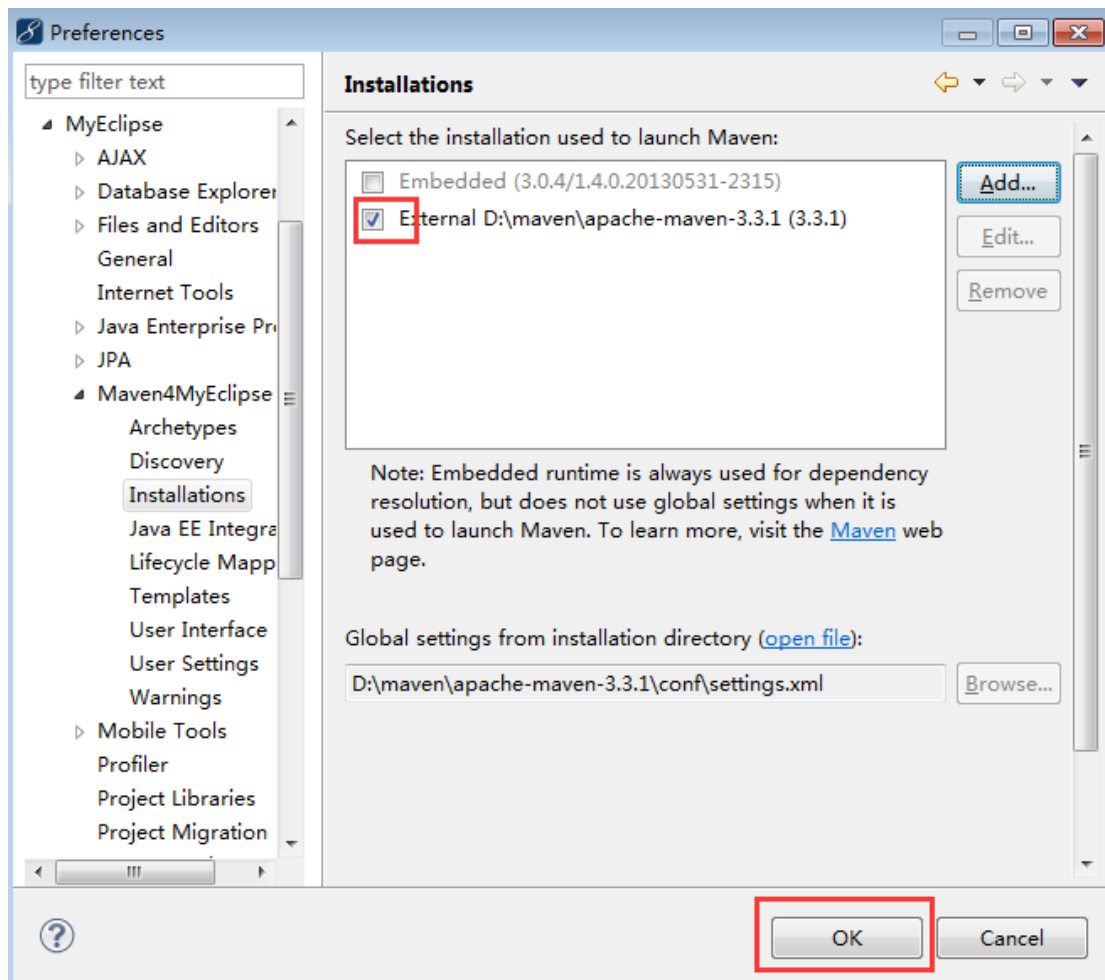
a) Window >> preferences >> MyEclipse >>Maven4MyEclipse >> Installations >> Add



b) 找到本地 Maven 工作目录：



c) 勾选刚添加的 maven , 点击 OK 按钮:

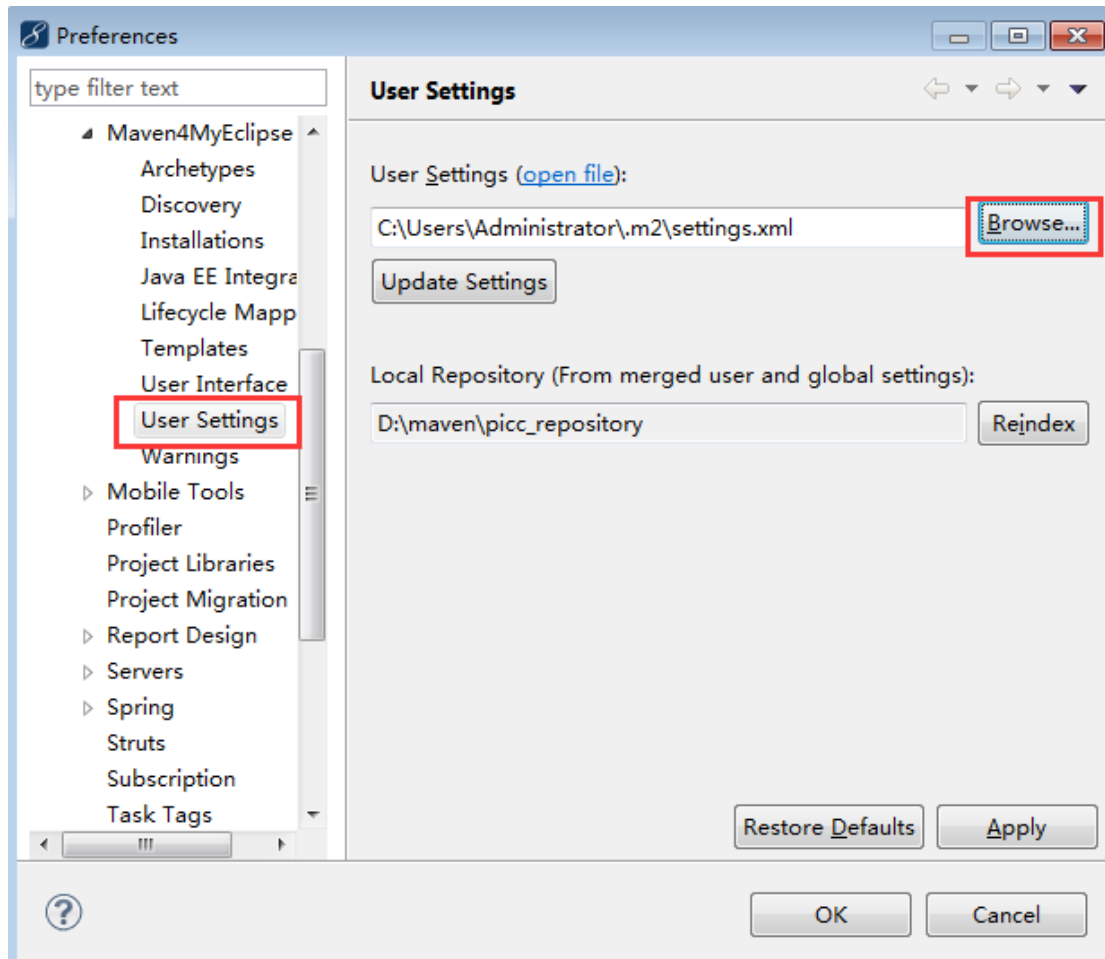


2 设置 User settings

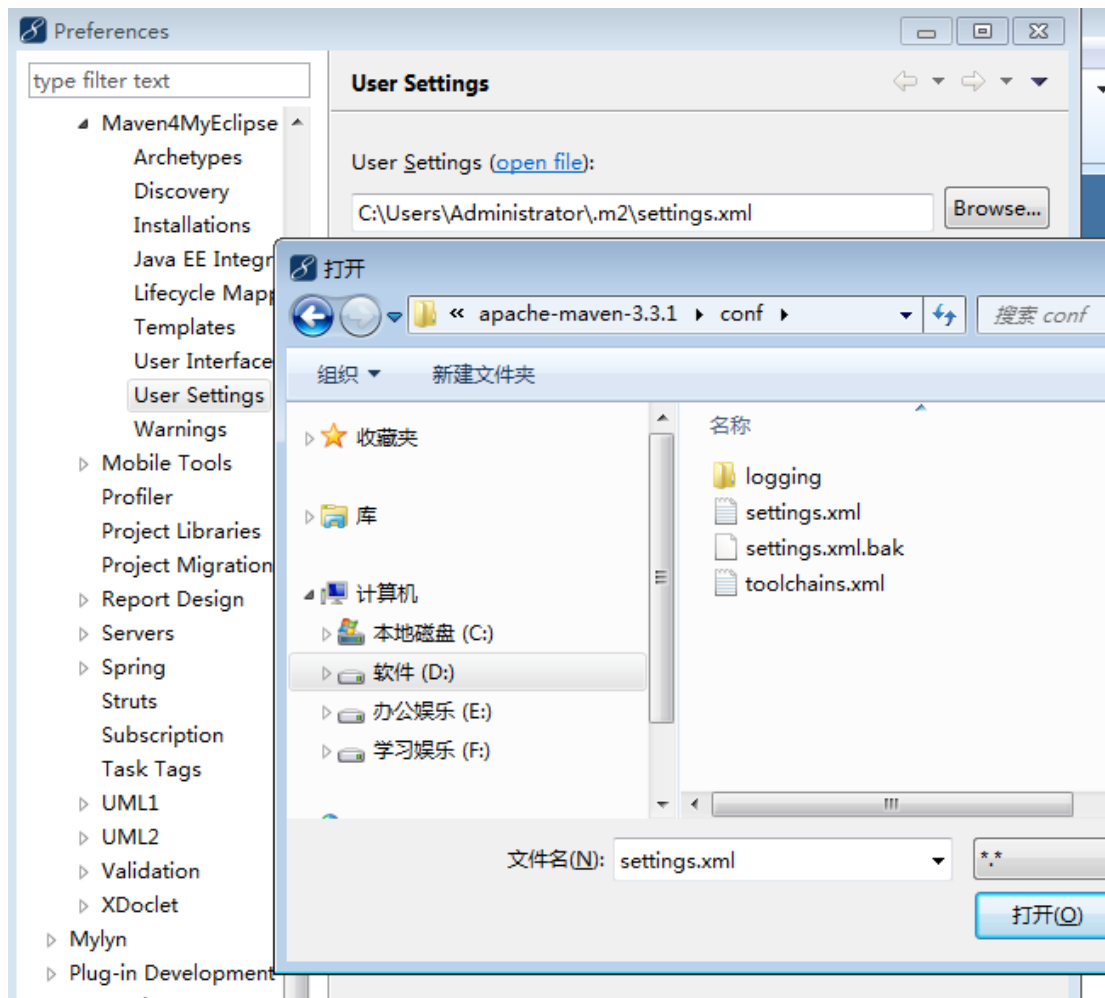
a) Window >> preferences >> MyEclipse >> Maven4MyEclipse

>> User Settings 如下图 , 分别将全局设置 Global Settings 和用户设置

User Settings, 都选为 maven 工作目录下面的 setting 文件

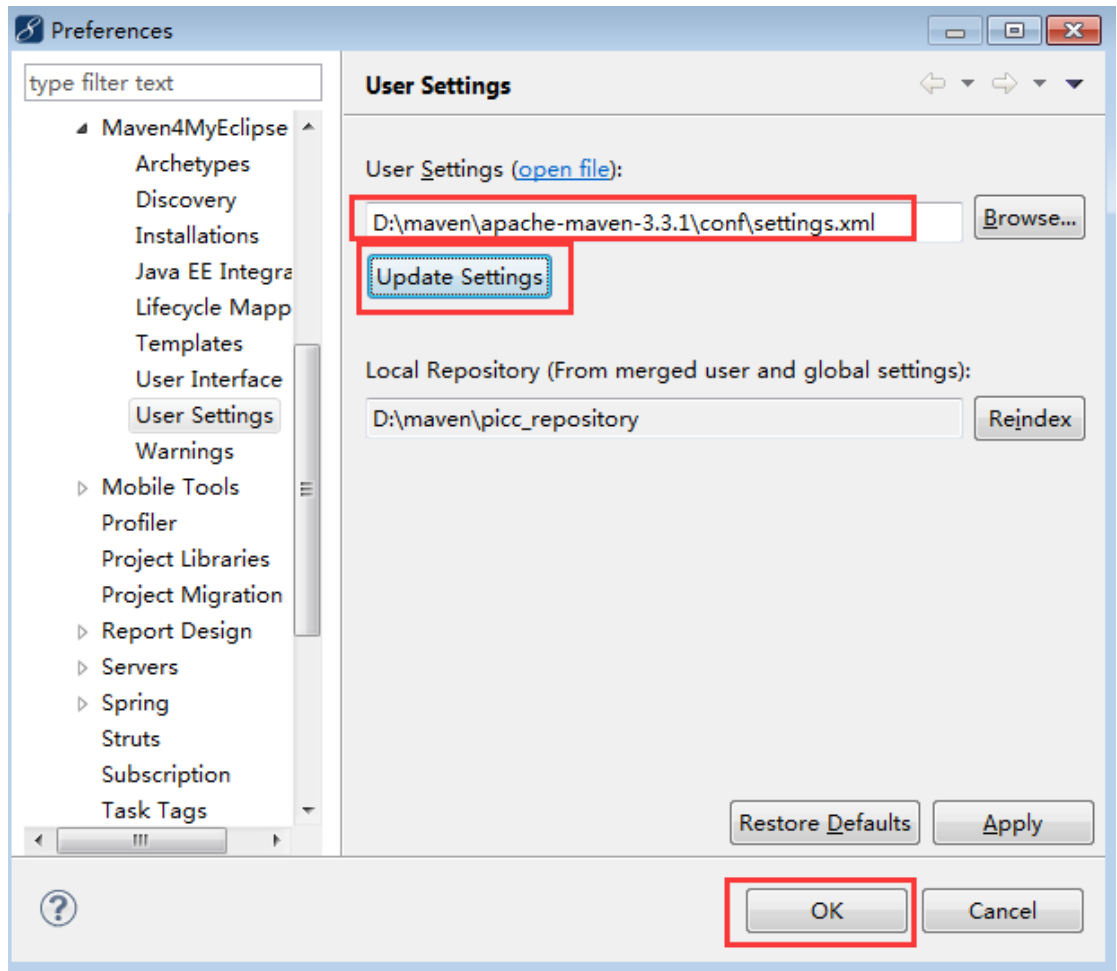


b) 点击 Browse



c) 点击 OK。

如果将来 settings 文件有修改，需要点击“更新 setting”。

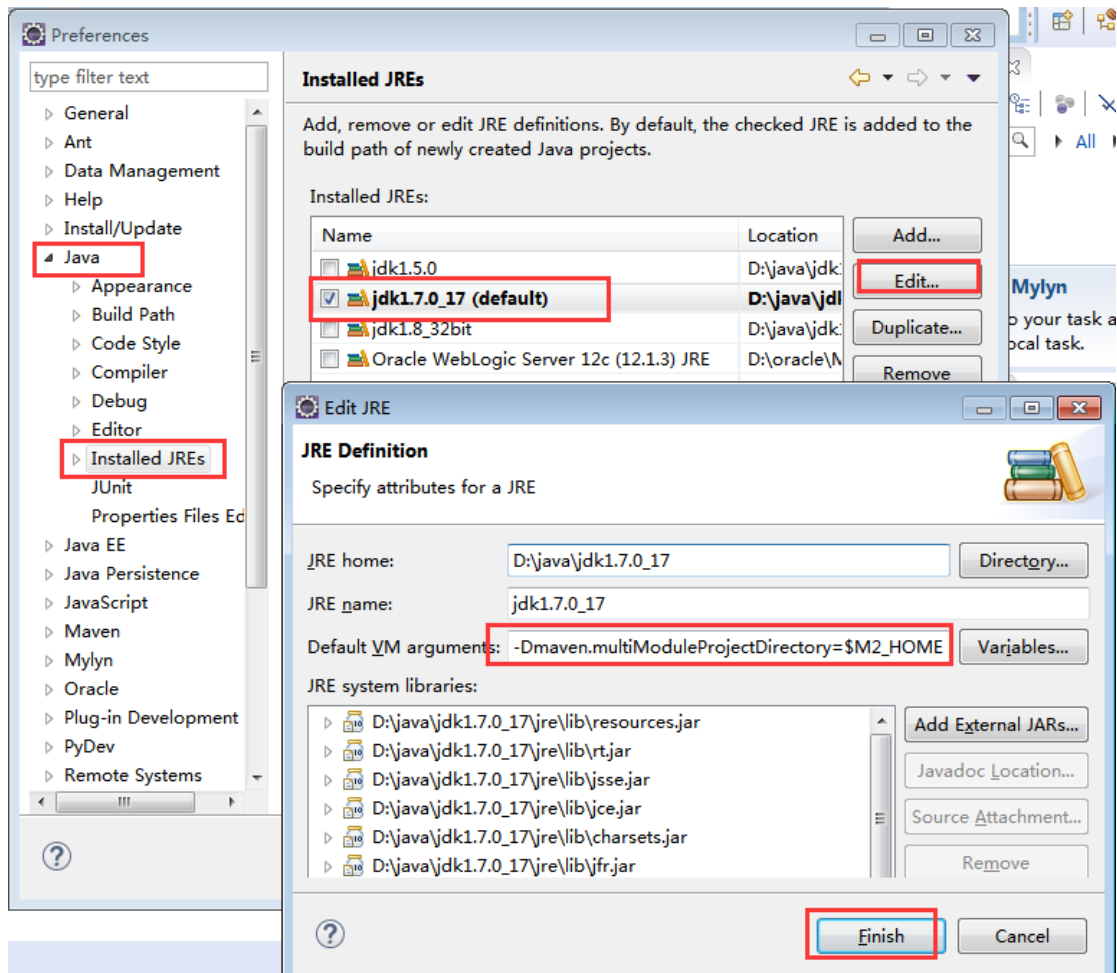


3 在 eclipse/Myeclipse 中配置 M2_HOME

为防止 eclipse 运行时找不到 maven 工作目录，需要在选择的 jre 中配置

-Dmaven.multiModuleProjectDirectory=\$M2_HOME,

以 jdk1.7 为例 Windows >> preferences >> java >> Installed JREs



点击 finish。

1.4 BeiMi 项目工程 Maven 化

1.4.1 Maven 标准目录结构说明

目录	说明
src	源码目录
src/main	主代码
src/main/java	主代码 java 文件
src/main/resources	主代码资源文件

src/test	测试代码
src/test/resources	测试代码资源文件
target	构建输出目录

1.4.2 pom.xml 详细配置

BeiMi 默认配置的 pom.xml 文件无需做修改。

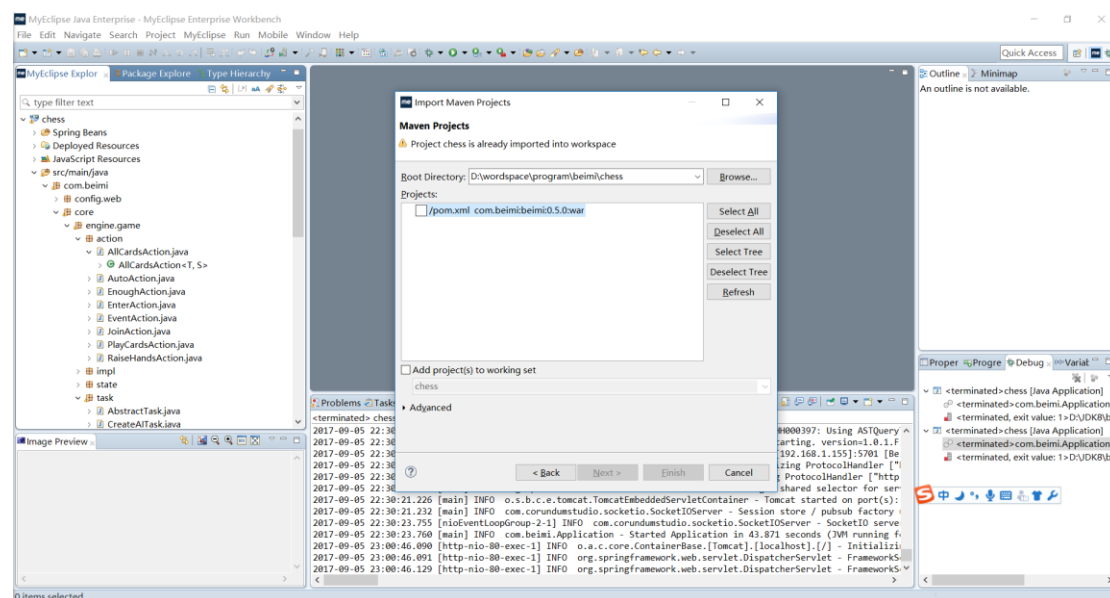
1.4.3 在 MyEclipse 中导入工程

a) 在导入前，做如下检查：

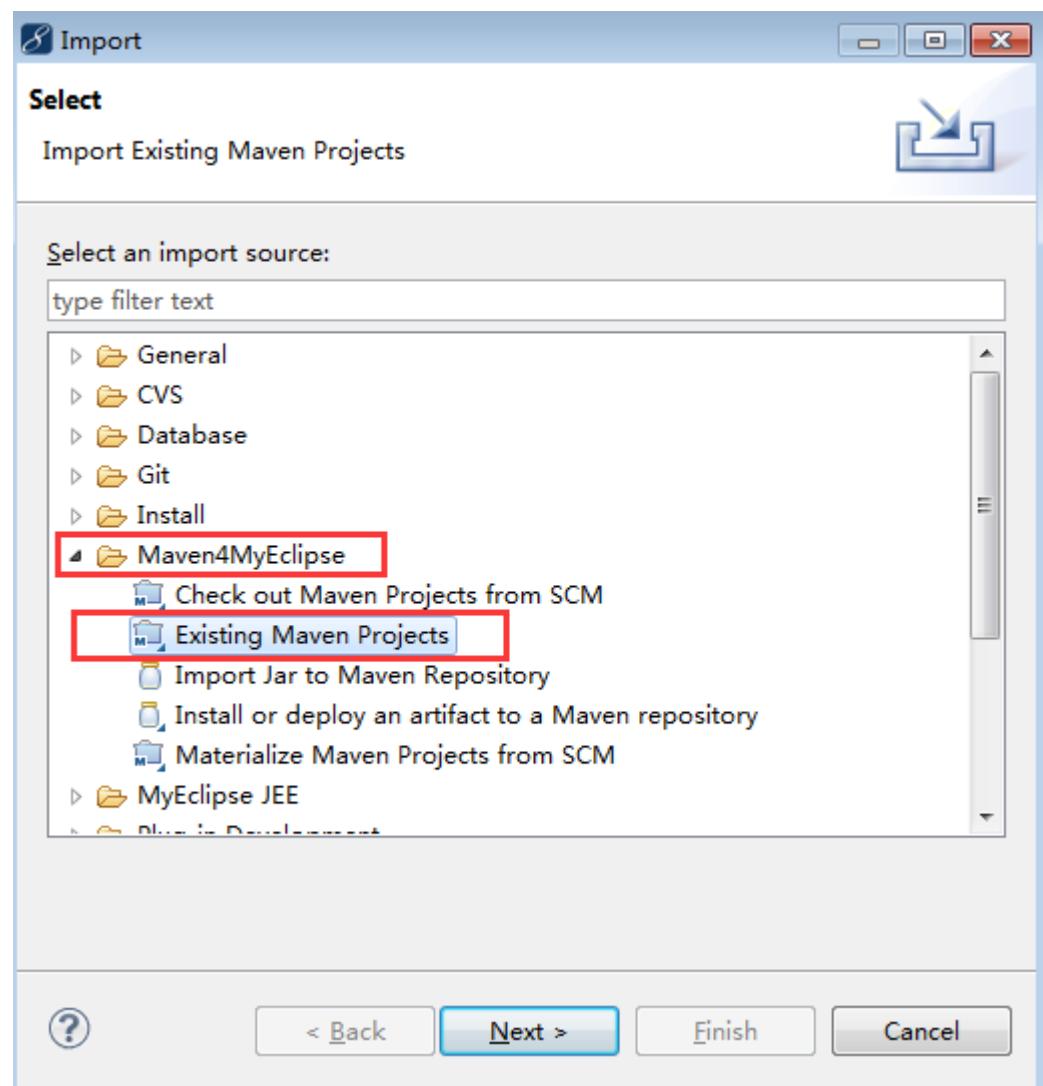
检查 settings.xml 中的阿里云仓库地址正确，并且是能联网的，同时，eclipse 中 maven 一定指定了我们自己的 settings.xml 文件（上面讲过如果配置），否则，导入项目是会报错：maven-compiler-plugin.jar 的问题。

检查 pom.xml 文件中的 groupId 和 name 要跟自己的项目保持一致：

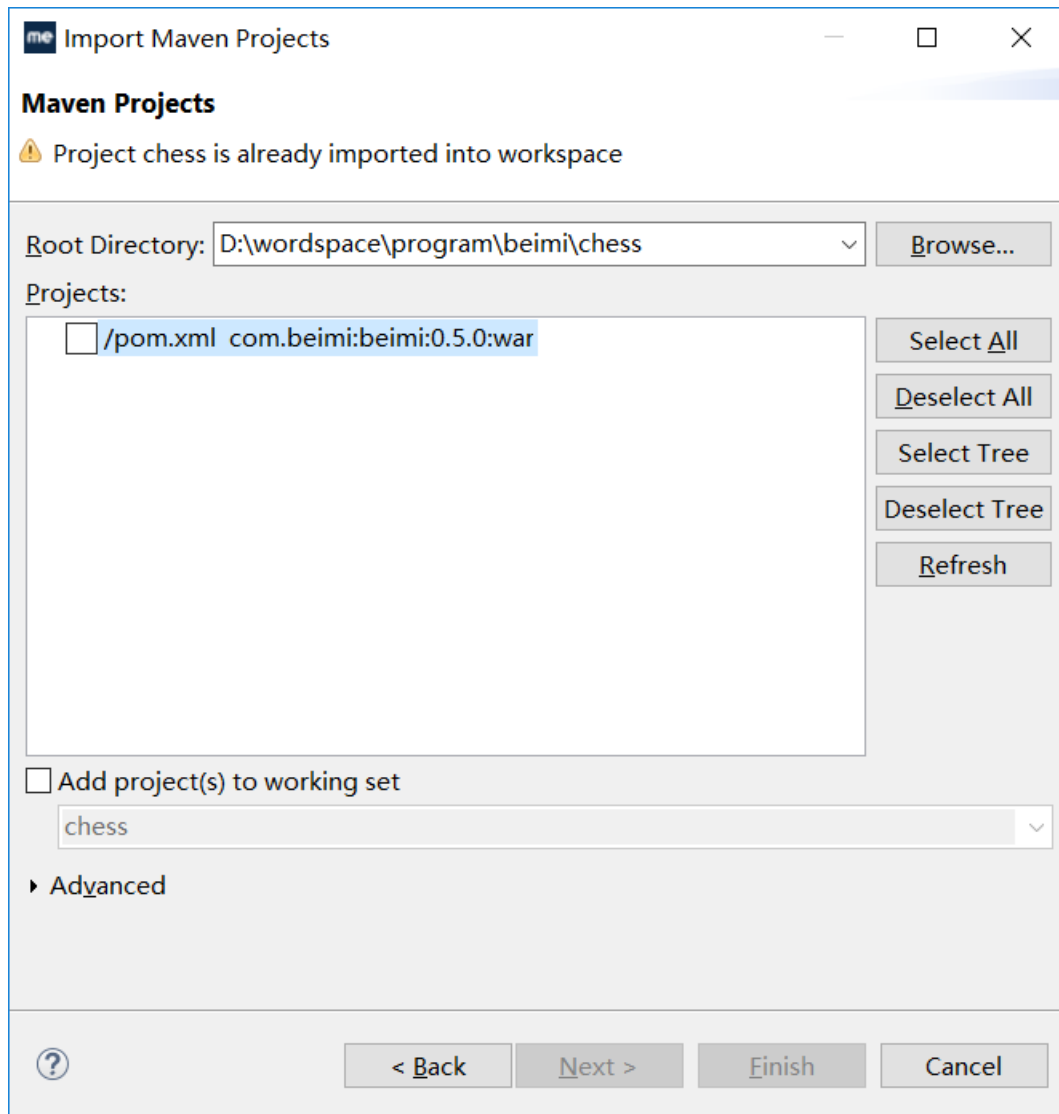
b) eclipse 导入工程时，file->import->maven,选择 Existing Maven Projects



Myeclipse 导入工程



下一步



- c) Next 后跟普通项目导入一样。
- d) 导入工程后 ,点击 Maven Console ,可以看到 maven 在按 pom.xml 中的配置自动下载 jar 文件 , 第一次下载时间比较长。

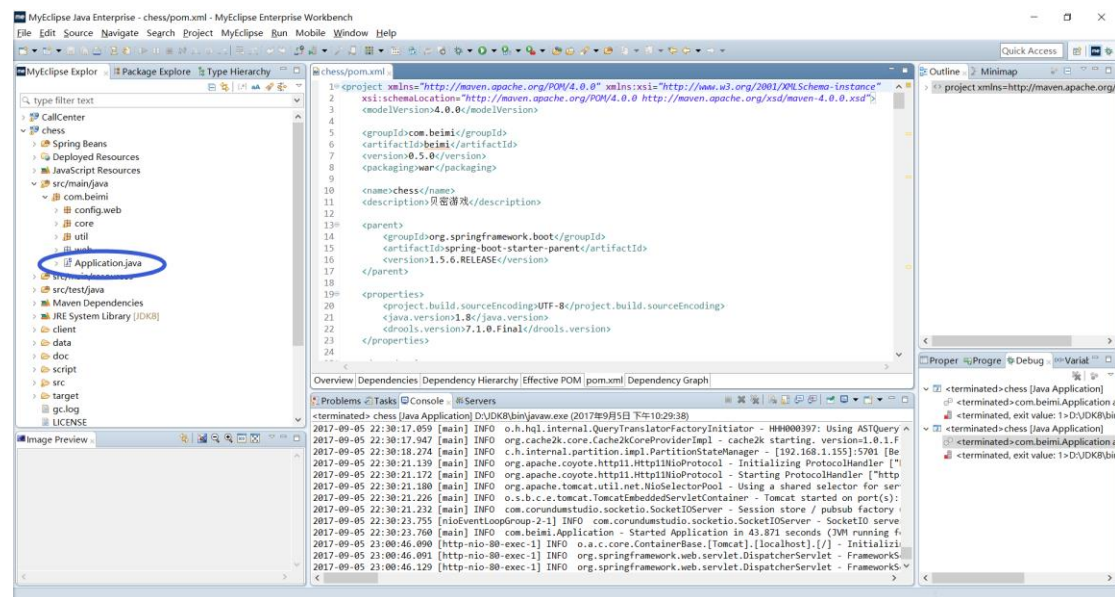
如果 jar 包更新有问题 , Problems 和 pom.xml 都会报错 , 按照错误提示修改 pom.xml 文件中的 jar 包依赖。

当我们修改 pom.xml 文件中的 jar 包依赖并保存时，eclipse 会自动下载 jar 包，如果不下下载，点中项目 >> 右键 >> Maven >> Update project(Alt+F5)。更多帮助，请参照某些 Jar 下载失败。

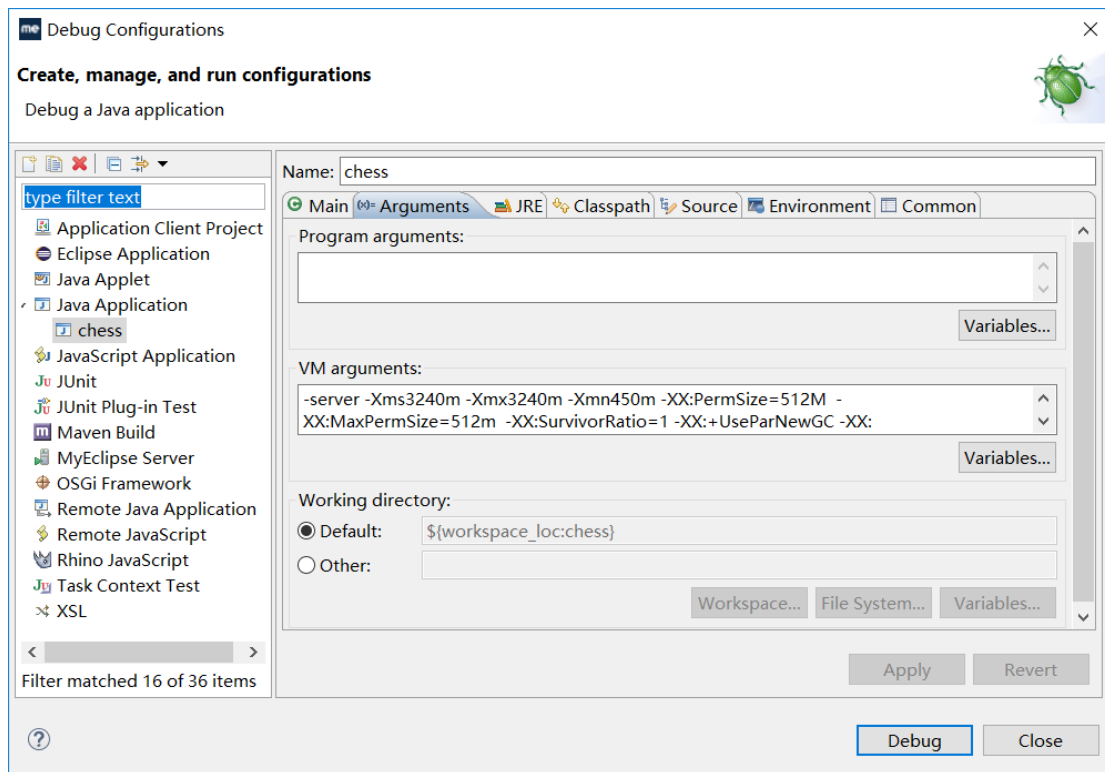
点击 OK,就会更新下载 jar。

1.4.4 运行 BeiMi 项目

BeiMi 在 MyEclipse 中可以直接运行，请在 BeiMi 工程的源代码目录中找到 Application.java



找到后右键->Run As -> Run Configurations...

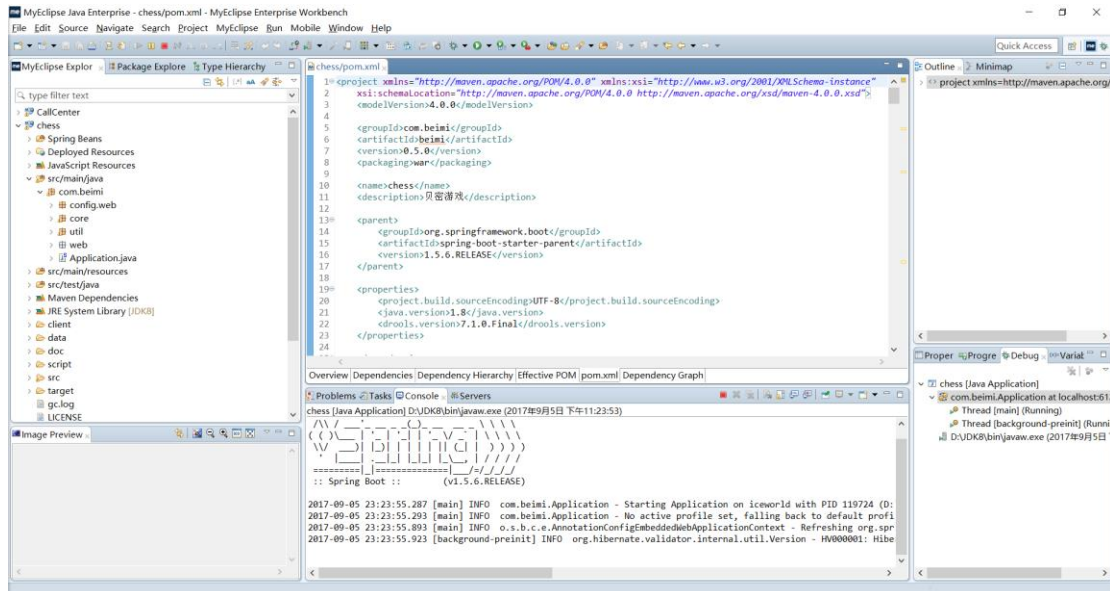


在 VM arguments 中增加如下参数

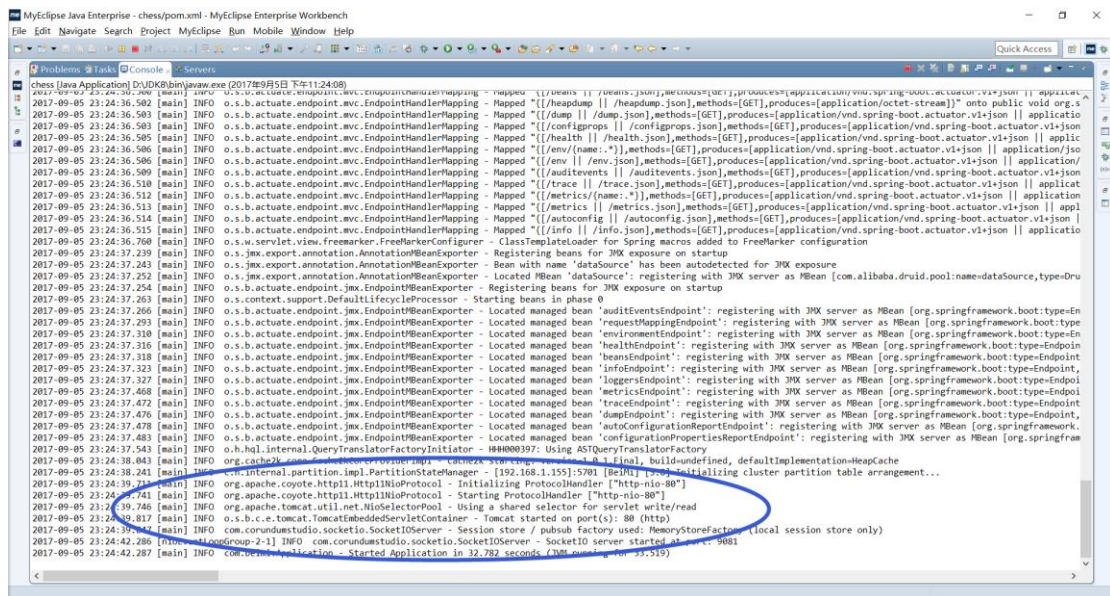
-Xms1240m -Xmx1240m -Xmn450m -XX:PermSize=512M
-XX:MaxPermSize=512m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC
-XX:+UseTLAB -XX:NewSize=128m -XX:MaxNewSize=128m
-XX:MaxTenuringThreshold=0 -XX:SurvivorRatio=1024
-XX:+UseCMSInitiatingOccupancyOnly
-XX:CMSInitiatingOccupancyFraction=60 -Djava.awt.headless=true
-XX:+PrintGCDetails -Xloggc:gc.log -XX:+PrintGCTimeStamps

点击 Apply 保存配置。

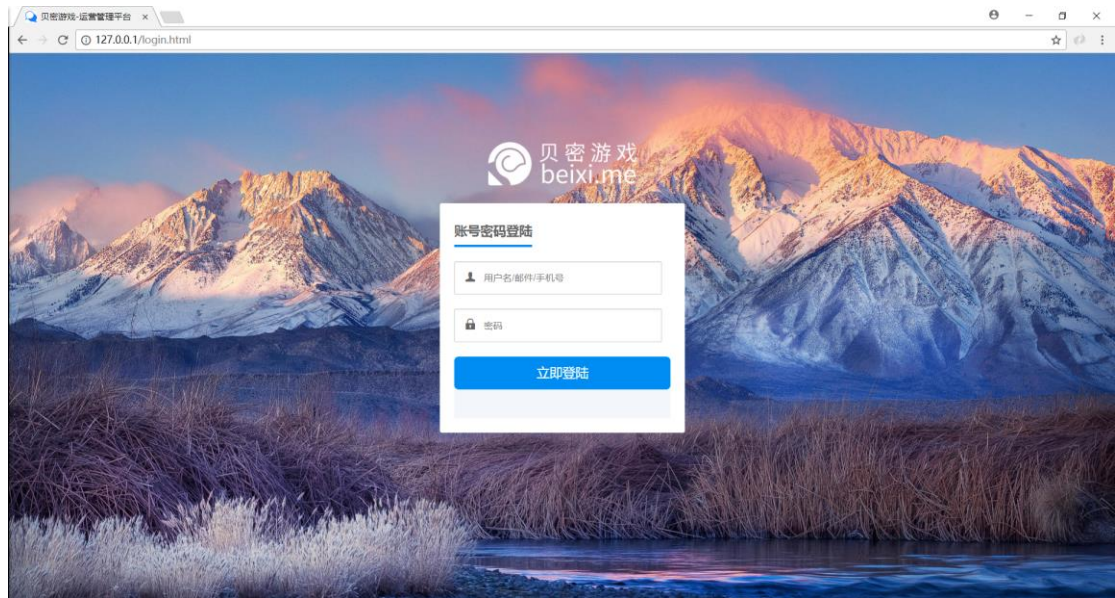
点击 Run , 看下控制台运行输出日志信息：



看到如下提示，即表示启动成功：



访问贝密游戏后台：



默认登陆账号和密码：admin/123456

2. 选读内容 (maven 使用配置及命令)

3.1 pom.xml 配置说明

配置详解:

```
<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/maven-v4_0_0.xsd">
```

<!--父项目的坐标。如果项目中没有规定某个元素的值，那么父项目中的对应值即为项目的默认值。 坐标包括 group ID，artifact ID 和 version。-->

```
<parent>
```

```
<!--被继承的父项目的构件标识符-->
```

```
<artifactId/>
```

```
<!--被继承的父项目的全球唯一标识符-->
```

```
<groupId/>
```

```
<!--被继承的父项目的版本-->
```

```
<version/>
```

<!--父项目的 pom.xml 文件的相对路径。相对路径允许你选择一个不同的路径。默认值是../pom.xml。Maven 首先在构建当前项目的地方寻找父项目的 pom，其次在文件系统的这个位置（relativePath 位置），然后在本地仓库，最后在远程仓库寻找父项目的 pom。-->

```
<relativePath/>
```

```
</parent>
```

<!--声明项目描述符遵循哪一个 POM 模型版本。模型本身的版本很少改变，虽然如此，但它仍然是必不可少的，这是为了当 Maven 引入了新的特性或者其他模型变更的时候，确保稳定性。-->

<modelVersion>4.0.0</modelVersion>

<!--项目的全球唯一标识符，通常使用全限定的包名区分该项目和其他项目。并且构建时生成的路径也是由此生成，如 com.mycompany.app 生成的相对路径为：/com/mycompany/app-->

<groupId>asia.banseon</groupId>

<!--构件的标识符，它和 group ID 一起唯一标识一个构件。换句话说，你不能有两个不同的项目拥有同样的 artifact ID 和 groupId；在某个特定的 group ID 下，artifact ID 也必须是唯一的。构件是项目产生的或使用的一个东西，Maven 为项目产生的构件包括：JARs，源码，二进制发布和 WARs 等。-->

<artifactId>banseon-maven2</artifactId>

<!--项目产生的构件类型，例如 jar、war、ear、pom。插件可以创建他们自己的构件类型，所以前面列的不是全部构件类型-->

<packaging>jar</packaging>

<!--项目当前版本，格式为:主版本.次版本.增量版本-限定版本号-->

<version>1.0-SNAPSHOT</version>

<!--项目的名称, Maven 产生的文档用-->

<name>banseon-maven</name>

<!--项目主页的 URL, Maven 产生的文档用-->

<url>http://www.baidu.com/banseon</url>

<!--项目的详细描述, Maven 产生的文档用。 当这个元素能够用 HTML 格式描述时(例如,CDATA 中的文本会被解析器忽略,就可以包含 HTML 标签), 不鼓励使用纯文本描述。如果你需要修改产生的 web 站点的索引页面,你应该修改你自己的索引页文件,而不是调整这里的文档。-->

<description>A maven project to study maven.</description>

<!--描述了这个项目构建环境中的前提条件。-->

<prerequisites>

<!--构建该项目或使用该插件所需要的 Maven 的最低版本-->

<maven/>

</prerequisites>

<!--项目的问题管理系统(Bugzilla, Jira, Scarab,或任何你喜欢的问题管理系统)的名称和 URL, 本例为 jira-->

<issueManagement>

<!--问题管理系统(例如 jira)的名字, -->

<system>jira</system>

<!--该项目使用的问题管理系统的 URL-->

<url>http://jira.baidu.com/banseon</url>

</issueManagement>

<!--项目持续集成信息-->

<ciManagement>

<!--持续集成系统的名字, 例如 continuum-->

<system/>

<!--该项目使用的持续集成系统的 URL (如果持续集成系统有 web 接口的话)。

-->

<url/>

<!--构建完成时，需要通知的开发者/用户的配置项。包括被通知者信息和通知条件（ 错误，失败，成功，警告 ）-->

<notifiers>

<!--配置一种方式，当构建中断时，以该方式通知用户/开发者-->

<notifier>

<!--传送通知的途径-->

<type/>

<!--发生错误时是否通知-->

<sendOnError/>

<!--构建失败时是否通知-->

<sendOnFailure/>

<!--构建成功时是否通知-->

<sendOnSuccess/>

<!--发生警告时是否通知-->

<sendOnWarning/>

<!--不赞成使用。通知发送到哪里-->

<address/>

<!--扩展配置项-->

<configuration/>

</notifier>

</notifiers>

</ciManagement>

<!--项目创建年份，4 位数字。当产生版权信息时需要使用这个值。-->

<inceptionYear/>

<!--项目相关邮件列表信息-->

<mailingLists>

<!--该元素描述了项目相关的所有邮件列表。自动产生的网站引用这些信息。-->

<mailingList>

<!--邮件的名称-->

<name>Demo</name>

<!--发送邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建-->

<post>banseon@126.com</post>

<!--订阅邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建-->

<subscribe>banseon@126.com</subscribe>

<!--取消订阅邮件的地址或链接，如果是邮件地址，创建文档时，mailto: 链接会被自动创建-->

<unsubscribe>banseon@126.com</unsubscribe>

<!--你可以浏览邮件信息的 URL-->

<archive>http://hi.baidu.com/banseon/demo/dev/</archive>

</mailingList>

</mailingLists>

<!--项目开发者列表-->

<developers>

<!--某个项目开发者的信息-->

<developer>

<!--SCM 里项目开发者的唯一标识符-->

<id>HELLO WORLD</id>

<!--项目开发者的全名-->

<name>banseon</name>

<!--项目开发者的 email-->

<email>banseon@126.com</email>

<!--项目开发者的主页的 URL-->

<url/>

<!--项目开发者在项目中扮演的角色，角色元素描述了各种角色-->

<roles>

<role>Project Manager</role>

<role>Architect</role>

</roles>

<!--项目开发者所属组织-->

<organization>demo</organization>

<!--项目开发者所属组织的 URL-->

<organizationUrl>http://hi.baidu.com/banseon</organizationUrl>

<!--项目开发者属性，如即时消息如何处理等-->

<properties>

<dept>No</dept>

</properties>

<!--项目开发者所在时区， -11 到 12 范围内的整数。-->

<timezone>-5</timezone>

</developer>

</developers>

<!--项目的其他贡献者列表-->

<contributors>

<!--项目的其他贡献者。参见 developers/developer 元素-->

<contributor>

<name/> <email/> <url/> <organization/> <organizationUrl/> <roles/> <time
zone/> <properties/>

</contributor>

</contributors>

<!--该元素描述了项目所有 License 列表。 应该只列出该项目的 license 列表，
不要列出依赖项目的 license 列表。如果列出多个 license，用户可以选择它们中
的一个而不是接受所有 license。-->

<licenses>

<!--描述了项目的 license，用于生成项目的 web 站点的 license 页面，其他一些

报表和 validation 也会用到该元素。-->

<license>

<!--license 用于法律上的名称-->

<name>Apache 2</name>

<!--官方的 license 正文页面的 URL-->

<url><http://www.baidu.com/banseon/LICENSE-2.0.txt></url>

<!--项目分发的主要方式：

repo，可以从 Maven 库下载

manual，用户必须手动下载和安装依赖-->

<distribution>repo</distribution>

<!--关于 license 的补充信息-->

<comments>A business-friendly OSS license</comments>

</license>

</licenses>

<!--SCM(Source Control Management)标签允许你配置你的代码库，供 Maven web 站点和其它插件使用。-->

<scm>

<!--SCM 的 URL,该 URL 描述了版本库和如何连接到版本库。欲知详情，请看 SCMs 提供的 URL 格式和列表。该连接只读。-->

<connection>

scm:svn:<http://svn.baidu.com/banseon/maven/banseon/banseon-maven2-trunk>(dao-trunk)

</connection>

<!--给开发者使用的，类似 connection 元素。即该连接不仅仅只读-->

<developerConnection>

scm:svn:http://svn.baidu.com/banseon/maven/banseon/dao-trunk

</developerConnection>

<!--当前代码的标签，在开发阶段默认为 HEAD-->

<tag/>

<!--指向项目的可浏览 SCM 库（例如 ViewVC 或者 Fisheye）的 URL。-->

<url>http://svn.baidu.com/banseon</url>

</scm>

<!--描述项目所属组织的各种属性。Maven 产生的文档用-->

<organization>

<!--组织的全名-->

<name>demo</name>

<!--组织主页的 URL-->

<url>http://www.baidu.com/banseon</url>

</organization>

<!--构建项目需要的信息-->

<build>

<!--该元素设置了项目源码目录，当构建项目的时候，构建系统会编译目录里的源码。该路径是相对于 pom.xml 的相对路径。-->

<sourceDirectory/>

<!--该元素设置了项目脚本源码目录,该目录和源码目录不同:绝大多数情况下,该目录下的内容 会被拷贝到输出目录(因为脚本是被解释的,而不是被编译的)。

-->

<scriptSourceDirectory/>

<!--该元素设置了项目单元测试使用的源码目录,当测试项目的时候,构建系统会编译目录里的源码。该路径是相对于 pom.xml 的相对路径。-->

<testSourceDirectory/>

<!--被编译过的应用程序 class 文件存放的目录。-->

<outputDirectory/>

<!--被编译过的测试 class 文件存放的目录。-->

<testOutputDirectory/>

<!--使用来自该项目的一系列构建扩展-->

<extensions>

<!--描述使用到的构建扩展。-->

<extension>

<!--构建扩展的 groupId-->

<groupId/>

<!--构建扩展的 artifactId-->

<artifactId/>

<!--构建扩展的版本-->

<version/>

</extension>

</extensions>

<!--当项目没有规定目标（Maven2 叫做阶段）时的默认值-->

<defaultGoal/>

<!--这个元素描述了项目相关的所有资源路径列表，例如和项目相关的属性文件，这些资源被包含在最终的打包文件里。-->

<resources>

<!--这个元素描述了项目相关或测试相关的所有资源路径-->

<resource>

<!-- 描述了资源的目标路径。该路径相对 target/classes 目录（例如 \${project.build.outputDirectory}）。举个例子，如果你想资源在特定的包里（org.apache.maven.messages），你必须该元素设置为 org/apache/maven/messages。然而，如果你只是想把资源放到源码目录结构里，就不需要该配置。-->

<targetPath/>

<!--是否使用参数值代替参数名。参数值取自 properties 元素或者文件里配置的属性，文件在 filters 元素里列出。-->

<filtering/>

<!--描述存放资源的目录，该路径相对 POM 路径-->

<directory/>

<!--包含的模式列表，例如**/*.xml.-->

<includes/>

<!--排除的模式列表，例如**/*.xml-->

<excludes/>

</resource>

</resources>

<!--这个元素描述了单元测试相关的所有资源路径，例如和单元测试相关的属性文件。-->

<testResources>

<!--这个元素描述了测试相关的所有资源路径，参见 build/resources/resource 元素的说明-->

<testResource>

<targetPath/> <filtering/> <directory/> <includes/> <excludes/>

</testResource>

</testResources>

<!--构建产生的所有文件存放的目录-->

<directory/>

<!--产生的构件的文件名，默认值是\${artifactId}-\${version}。-->

<finalName/>

<!--当 filtering 开关打开时，使用到的过滤器属性文件列表-->

<filters/>

<!--子项目可以引用的默认插件信息。该插件配置项直到被引用时才会被解析或绑定到生命周期。给定插件的任何本地配置都会覆盖这里的配置-->

<pluginManagement>

<!--使用的插件列表。-->

<plugins>

<!--plugin 元素包含描述插件所需要的信息。-->

<plugin>

<!--插件在仓库里的 group ID-->

<groupId/>

<!--插件在仓库里的 artifact ID-->

<artifactId/>

<!--被使用的插件的版本（或版本范围）-->

<version/>

<!--是否从该插件下载 Maven 扩展（例如打包和类型处理器），由于性能原因，只有在真需要下载时，该元素才被设置成 enabled。-->

<extensions/>

<!--在构建生命周期中执行一组目标的配置。每个目标可能有不同的配置。-->

<executions>

<!--execution 元素包含了插件执行需要的信息-->

<execution>

<!--执行目标的标识符，用于标识构建过程中的目标，或者匹配继承过程中需要合并的执行目标-->

<id/>

<!--绑定了目标的构建生命周期阶段，如果省略，目标会被绑定到源数据里配置的默认阶段-->

<phase/>

```
<!--配置的执行目标-->

<goals/>

<!--配置是否被传播到子 POM-->

<inherited/>

<!--作为 DOM 对象的配置-->

<configuration/>

</execution>

</executions>

<!--项目引入插件所需要的额外依赖-->

<dependencies>

<!--参见 dependencies/dependency 元素-->

<dependency>

.....

</dependency>

</dependencies>

<!--任何配置是否被传播到子项目-->

<inherited/>

<!--作为 DOM 对象的配置-->

<configuration/>

</plugin>

</plugins>

</pluginManagement>
```

<!--使用的插件列表-->

<plugins>

<!--参见 build/pluginManagement/plugins/plugin 元素-->

<plugin>

<groupId/> <artifactId/> <version/> <extensions/>

<executions>

<execution>

<id/> <phase/> <goals/> <inherited/> <configuration/>

</execution>

</executions>

<dependencies>

<!--参见 dependencies/dependency 元素-->

<dependency>

.....

</dependency>

</dependencies>

<goals/> <inherited/> <configuration/>

</plugin>

</plugins>

</build>

<!--在列的项目构建 profile，如果被激活，会修改构建处理-->

<profiles>

<!--根据环境参数或命令行参数激活某个构建处理-->

<profile>

<!--构建配置的唯一标识符。即用于命令行激活，也用于在继承时合并具有相同标识符的 profile。-->

<id/>

<!--自动触发 profile 的条件逻辑。Activation 是 profile 的开启钥匙。profile 的力量来自于它

能够在某些特定的环境中自动使用某些特定的值；这些环境通过 activation 元素指定。activation 元素并不是激活 profile 的唯一方式。-->

<activation>

<!--profile 默认是否激活的标志-->

<activeByDefault/>

<!--当匹配的 jdk 被检测到，profile 被激活。例如，1.4 激活 JDK1.4，1.4.0_2，而!1.4 激活所有版本不是以 1.4 开头的 JDK。-->

<jdk/>

<!--当匹配的操作系统属性被检测到，profile 被激活。os 元素可以定义一些操作系统相关的属性。-->

<os>

<!--激活 profile 的操作系统的名字-->

<name>Windows XP</name>

<!--激活 profile 的操作系统所属家族(如 'windows')-->

<family>Windows</family>

<!--激活 profile 的操作系统体系结构 -->

<arch>x86</arch>

<!--激活 profile 的操作系统版本-->

<version>5.1.2600</version>

</os>

<!--如果 Maven 检测到某一个属性（其值可以在 POM 中通过\${名称}引用），其拥有对应的名称和值，Profile 就会被激活。如果值

字段是空的，那么存在属性名称字段就会激活 profile，否则按区分大小写方式匹配属性值字段-->

<property>

<!--激活 profile 的属性的名称-->

<name>mavenVersion</name>

<!--激活 profile 的属性的值-->

<value>2.0.3</value>

</property>

<!--提供一个文件名，通过检测该文件的存在或不存在来激活 profile。missing 检查文件是否存在，如果不存在则激活

profile。另一方面，exists 则会检查文件是否存在，如果存在则激活 profile。-->

<file>

<!--如果指定的文件存在，则激活 profile。-->

<exists>/usr/local/hudson/hudson-home/jobs/maven-guide-zh-to-producti
on/workspace/</exists>

```
<!--如果指定的文件不存在，则激活 profile。-->

<missing>/usr/local/hudson/hudson-home/jobs/maven-guide-zh-to-produ
ction/workspace/</missing>

</file>

</activation>

<!--构建项目所需要的信息。参见 build 元素-->

<build>

<defaultGoal/>

<resources>

<resource>

<targetPath/> <filtering/> <directory/> <includes/> <excludes/>

</resource>

</resources>

<testResources>

<testResource>

<targetPath/> <filtering/> <directory/> <includes/> <excludes/>

</testResource>

</testResources>

<directory/> <finalName/> <filters/>

<pluginManagement>

<plugins>

<!--参见 build/pluginManagement/plugins/plugin 元素-->
```

```
<plugin>

<groupId/> <artifactId/> <version/> <extensions/>

<executions>

<execution>

<id/> <phase/> <goals/> <inherited/> <configuration/>

</execution>

</executions>

<dependencies>

<!--参见 dependencies/dependency 元素-->

<dependency>

.....

</dependency>

</dependencies>

<goals/> <inherited/> <configuration/>

</plugin>

</plugins>

</pluginManagement>

<plugins>

<!--参见 build/pluginManagement/plugins/plugin 元素-->

<plugin>

<groupId/> <artifactId/> <version/> <extensions/>

<executions>
```

<execution>

<id/> <phase/> <goals/> <inherited/> <configuration/>

</execution>

</executions>

<dependencies>

<!--参见 dependencies/dependency 元素-->

<dependency>

.....

</dependency>

</dependencies>

<goals/> <inherited/> <configuration/>

</plugin>

</plugins>

</build>

<!--模块（有时称作子项目）被构建成项目的一部分。列出的每个模块元素是指向该模块的目录的相对路径-->

<modules/>

<!--发现依赖和扩展的远程仓库列表。-->

<repositories>

<!--参见 repositories/repository 元素-->

<repository>

<releases>

```
<enabled/> <updatePolicy/> <checksumPolicy/>

</releases>

<snapshots>

<enabled/> <updatePolicy/> <checksumPolicy/>

</snapshots>

<id/> <name/> <url/> <layout/>

</repository>

</repositories>

<!--发现插件的远程仓库列表，这些插件用于构建和报表-->

<pluginRepositories>

<!--包含需要连接到远程插件仓库的信息.参见 repositories/repository 元素-->

<pluginRepository>

<releases>

<enabled/> <updatePolicy/> <checksumPolicy/>

</releases>

<snapshots>

<enabled/> <updatePolicy/> <checksumPolicy/>

</snapshots>

<id/> <name/> <url/> <layout/>

</pluginRepository>

</pluginRepositories>

<!--该元素描述了项目相关的所有依赖。 这些依赖组成了项目构建过程中的一
```

各个环节。它们自动从项目定义的仓库中下载。要获取更多信息，请看项目依赖机制。-->

```
<dependencies>
```

```
<!--参见 dependencies/dependency 元素-->
```

```
<dependency>
```

```
.....
```

```
</dependency>
```

```
</dependencies>
```

```
<!--不赞成使用. 现在 Maven 忽略该元素.-->
```

```
<reports/>
```

```
<!--该元素包括使用报表插件产生报表的规范。当用户执行“mvn site”，这些报表就会运行。 在页面导航栏能看到所有报表的链接。参见 reporting 元素-->
```

```
<reporting>
```

```
.....
```

```
</reporting>
```

```
<!--参见 dependencyManagement 元素-->
```

```
<dependencyManagement>
```

```
<dependencies>
```

```
<!--参见 dependencies/dependency 元素-->
```

```
<dependency>
```

```
.....
```

```
</dependency>
```

</dependencies>

</dependencyManagement>

<!--参见 distributionManagement 元素-->

<distributionManagement>

.....

</distributionManagement>

<!--参见 properties 元素-->

<properties/>

</profile>

</profiles>

<!--模块（有时称作子项目）被构建成项目的一部分。列出的每个模块元素是指向该模块的目录的相对路径-->

<modules/>

<!--发现依赖和扩展的远程仓库列表。-->

<repositories>

<!--包含需要连接到远程仓库的信息-->

<repository>

<!--如何处理远程仓库里发布版本的下载-->

<releases>

<!--true 或者 false 表示该仓库是否为下载某种类型构件（发布版，快照版）开启。-->

<enabled/>

<!--该元素指定更新发生的频率。Maven 会比较本地 POM 和远程 POM 的时间戳。这里的选项是：always（一直），daily（默认，每日），interval：X（这里 X 是以分钟为单位的时间间隔），或者 never（从不）。-->

<updatePolicy/>

<!--当 Maven 验证构件校验文件失败时该怎么做：ignore（忽略），fail（失败），或者 warn（警告）。-->

<checksumPolicy/>

</releases>

<!--如何处理远程仓库里快照版本的下载。有了 releases 和 snapshots 这两组配置，POM 就可以在每个单独的仓库中，为每种类型的构件采取不同的策略。例如，可能有人会决定只为开发目的开启对快照版本下载的支持。参见 repositories/repository/releases 元素-->

<snapshots>

<enabled/> <updatePolicy/> <checksumPolicy/>

</snapshots>

<!--远程仓库唯一标识符。可以用来匹配在 settings.xml 文件里配置的远程仓库-->

<id>banseon-repository-proxy</id>

<!--远程仓库名称-->

<name>banseon-repository-proxy</name>

<!--远程仓库 URL，按 protocol://hostname/path 形式-->

<url>http://192.168.1.169:9999/repository/</url>

<!--用于定位和排序构件的仓库布局类型-可以是 default(默认)或者 legacy(遗留)。Maven 2 为其仓库提供了一个默认的布局；然而，Maven 1.x 有一种不同的布局。我们可以使用该元素指定布局是 default (默认) 还是 legacy (遗留)。

-->

<layout>default</layout>

</repository>

</repositories>

<!--发现插件的远程仓库列表，这些插件用于构建和报表-->

<pluginRepositories>

<!--包含需要连接到远程插件仓库的信息.参见 repositories/repository 元素-->

<pluginRepository>

.....

</pluginRepository>

</pluginRepositories>

<!--该元素描述了项目相关的所有依赖。 这些依赖组成了项目构建过程中的一个个环节。它们自动从项目定义的仓库中下载。要获取更多信息，请看项目依赖机制。 -->

<dependencies>

<dependency>

<!--依赖的 group ID-->

<groupId>org.apache.maven</groupId>

<!-- 依赖的 artifact ID -->

<artifactId>maven-artifact</artifactId>

<!-- 依赖的版本号。 在 Maven 2 里, 也可以配置成版本号的范围。 -->

<version>3.8.1</version>

<!-- 依赖类型, 默认类型是 jar。它通常表示依赖的文件的扩展名, 但也有例外。

一个类型可以被映射成另外一个扩展名或分类器。类型经常和使用的打包方式对应, 尽管这也有例外。一些类型的例子: jar, war, ejb-client 和 test-jar。如果设置 extensions 为 true, 就可以在 plugin 里定义新的类型。所以前面的类型的例子不完整。 -->

<type>jar</type>

<!-- 依赖的分类器。分类器可以区分属于同一个 POM, 但不同构建方式的构件。

分类器名被附加到文件名的版本号后面。例如, 如果你想要构建两个单独的构件成 JAR, 一个使用 Java 1.4 编译器, 另一个使用 Java 6 编译器, 你就可以使用分类器来生成两个单独的 JAR 构件。 -->

<classifier></classifier>

<!-- 依赖范围。在项目发布过程中, 帮助决定哪些构件被包括进来。欲知详情请参考依赖机制。

- compile : 默认范围, 用于编译
- provided : 类似于编译, 但支持你期待 jdk 或者容器提供, 类似于 classpath
- runtime: 在执行时需要使用
- test: 用于 test 任务时使用
- system: 需要外在提供相应的元素。通过 systemPath 来取得

- systemPath: 仅用于范围为 system。提供相应的路径

- optional: 当项目自身被依赖时，标注依赖是否传递。用于连续依赖时使用-->

```
<scope>test</scope>
```

<!--仅供 system 范围使用。注意，不鼓励使用这个元素，并且在新的版本中该元素可能被覆盖掉。该元素为依赖规定了文件系统上的路径。需要绝对路径而不是相对路径。推荐使用属性匹配绝对路径，例如\${java.home}。-->

```
<systemPath></systemPath>
```

<!--当计算传递依赖时，从依赖构件列表里，列出被排除的依赖构件集。即告诉 maven 你只依赖指定的项目，不依赖项目的依赖。此元素主要用于解决版本冲突问题-->

```
<exclusions>
```

```
<exclusion>
```

```
<artifactId>spring-core</artifactId>
```

```
<groupId>org.springframework</groupId>
```

```
</exclusion>
```

```
</exclusions>
```

<!--可选依赖，如果你在项目 B 中把 C 依赖声明为可选，你就需要在依赖于 B 的项目（例如项目 A）中显式的引用对 C 的依赖。可选依赖阻断依赖的传递性。-->

```
<optional>true</optional>
```

```
</dependency>
```

```
</dependencies>
```

<!--不赞成使用. 现在 Maven 忽略该元素.-->

<reports></reports>

<!--该元素描述使用报表插件产生报表的规范。当用户执行“mvn site”，这些报表就会运行。 在页面导航栏能看到所有报表的链接。-->

<reporting>

<!--true，则，网站不包括默认的报表。这包括“项目信息”菜单中的报表。-->

<excludeDefaults/>

<!--所有产生的报表存放到哪里。默认值是\${project.build.directory}/site。-->

<outputDirectory/>

<!--使用的报表插件和他们的配置。-->

<plugins>

<!--plugin 元素包含描述报表插件需要的信息-->

<plugin>

<!--报表插件在仓库里的 group ID-->

<groupId/>

<!--报表插件在仓库里的 artifact ID-->

<artifactId/>

<!--被使用的报表插件的版本（或版本范围）-->

<version/>

<!--任何配置是否被传播到子项目-->

<inherited/>

<!--报表插件的配置-->

<configuration/>

<!--一组报表的多重规范，每个规范可能有不同的配置。一个规范（报表集）对应一个执行目标。例如，有 1, 2, 3, 4, 5, 6, 7, 8, 9 个报表。1, 2, 5 构成 A 报表集，对应一个执行目标。2, 5, 8 构成 B 报表集，对应另一个执行目标-->

<reportSets>

<!--表示报表的一个集合，以及产生该集合的配置-->

<reportSet>

<!--报表集合的唯一标识符，POM 继承时用到-->

<id/>

<!--产生报表集合时，被使用的报表的配置-->

<configuration/>

<!--配置是否被继承到子 POMs-->

<inherited/>

<!--这个集合里使用到哪些报表-->

<reports/>

</reportSet>

</reportSets>

</plugin>

</plugins>

</reporting>

<!--继承自该项目的所有子项目的默认依赖信息。这部分的依赖信息不会被立即

解析,而是当子项目声明一个依赖（必须描述 group ID 和 artifact ID 信息）, 如果 group ID 和 artifact ID 以外的一些信息没有描述, 则通过 group ID 和 artifact ID 匹配到这里的依赖, 并使用这里的依赖信息。-->

```
<dependencyManagement>
```

```
<dependencies>
```

```
<!--参见 dependencies/dependency 元素-->
```

```
<dependency>
```

```
.....
```

```
</dependency>
```

```
</dependencies>
```

```
</dependencyManagement>
```

<!--项目分发信息, 在执行 mvn deploy 后表示要发布的位置。有了这些信息就可以把网站部署到远程服务器或者把构件部署到远程仓库。-->

```
<distributionManagement>
```

```
<!--部署项目产生的构件到远程仓库需要的信息-->
```

```
<repository>
```

<!--是分配给快照一个唯一的版本号（由时间戳和构建流水号）？还是每次都使用相同的版本号？参见 repositories/repository 元素-->

```
<uniqueVersion/>
```

```
<id>banseon-maven2</id>
```

```
<name>banseon maven2</name>
```

```
<url>file://${basedir}/target/deploy</url>
```

<layout/>

</repository>

<!--构件的快照部署到哪里？如果没有配置该元素，默认部署到 repository 元素配置的仓库，参见 distributionManagement/repository 元素-->

<snapshotRepository>

<uniqueVersion/>

<id>banseon-maven2</id>

<name>Banseon-maven2 Snapshot Repository</name>

<url>scp://svn.baidu.com/banseon:/usr/local/maven-snapshot</url>

<layout/>

</snapshotRepository>

<!--部署项目的网站需要的信息-->

<site>

<!--部署位置的唯一标识符，用来匹配站点和 settings.xml 文件里的配置-->

<id>banseon-site</id>

<!--部署位置的名称-->

<name>business api website</name>

<!--部署位置的 URL，按 protocol://hostname/path 形式-->

<url>

scp://svn.baidu.com/banseon:/var/www/localhost/banseon-web

</url>

</site>

<!--项目下载页面的 URL。如果没有该元素，用户应该参考主页。使用该元素的原因是：帮助定位那些不在仓库里的构件（由于 license 限制）。-->

<downloadUrl/>

<!--如果构件有了新的 group ID 和 artifact ID（构件移到了新的位置），这里列出构件的重定位信息。-->

<relocation>

<!--构件新的 group ID-->

<groupId/>

<!--构件新的 artifact ID-->

<artifactId/>

<!--构件新的版本号-->

<version/>

<!--显示给用户的，关于移动的额外信息，例如原因。-->

<message/>

</relocation>

<!--给出该构件在远程仓库的状态。不得在本地项目中设置该元素，因为这是工具自动更新的。有效的值有：none（默认），converted（仓库管理员从 Maven 1 POM 转换过来），partner（直接从伙伴 Maven 2 仓库同步过来），deployed（从 Maven 2 实例部署），verified（被核实时正确的和最终的）。-->

<status/>

</distributionManagement>

<!--以值替代名称，Properties 可以在整个 POM 中使用，也可以作为触发条件

(见 settings.xml 配置文件里 activation 元素的说明) 。格式是

```
<name>value</name>。-->
```

```
<properties/>
```

```
</project>
```

3.2 maven 命令使用

```
mvn help:describe
```

你是否因为记不清某个插件有哪些 goal 而痛苦过,你是否因为想不起某个 goal 有哪些参数而苦恼,那就试试这个命令吧,它会告诉你一切的.

参数: 1. -Dplugin=pluginName 2. -Dgoal(或-Dmojo)=goalName:与-Dplugin 一起使用,它会列出某个插件的 goal 信息,

如果嫌不够详细,同样可以加-Ddetail.(注:一个插件 goal 也被认为是一个 "Mojo")

下面大家就运行 mvn help:describe -Dplugin=help -Dmojo=describe 感受一下 吧!

```
mvn archetype:generate
```

你是怎么创建你的 maven 项目的?是不是像这样:mvn archetype:create -DarchetypeArtifactId=maven-archetype-quickstart -DgroupId=com.ryanote -Dartifact=common,

如果你还再用的话,那你就 out 了,现代人都用 `mvn archetype:generate` 了,它将创建项目这件枯燥的事更加人性化,你再也不需要记那么多的 `archetypeArtifactId`,你只需输入 `archetype:generate`,剩下的就是做“选择题”了.

`mvn tomcat:run`

用了 maven 后,你再也不需要用 eclipse 里的 tomcat 来运行 web 项目(实际工作中经常会发现用它会出现不同步更新的情况),只需在对应目录里运行 `mvn tomcat:run` 命令,

然后就可在浏览器里运行查看了.如果你想要更多的定制,可以在 `pom.xml` 文件里加下面配置:

01 02 03 04 `org.codehaus.mojo` 05 `tomcat-maven-plugin` 06 07 `/web` 08 9090
09 10 11 12 当然你也可以在命令里加参数来实现特定的功能,

下面几个比较常用:

1. 跳过测试:`-Dmaven.test.skip(=true)`
2. 指定端口:`-Dmaven.tomcat.port=9090`
3. 忽略测试失败:`-Dmaven.test.failure.ignore=true` 当然,如果你的其它关联项目有过更新的话,一定要在项目根目录下运行 `mvn clean install` 来执行更新,再运行 `mvn tomcat:run` 使改动生效.
4. `mvnDebug tomcat:run`

这条命令主要用来远程测试,它会监听远程测试用的 8000 端口,在 eclipse 里打开远程测试后,它就会跑起来了,设断点,调试,一切都是这么简单.上面提到的那几个参数在这里同样适用.

5. mvn dependency:sources

故名思义,有了它,你就不用到处找源码了,运行一下,你项目里所依赖的 jar 包的源码就都有了

Maven 常用命令：

1. 创建 Maven 的普通 java 项目：

```
mvn archetype:create -DgroupId=packageName -DartifactId=projectName
```

2. 创建 Maven 的 Web 项目：

```
mvn archetype:create -DgroupId=packageName -DartifactId=webappName -DarchetypeArtifactId=maven-archetype-webapp
```

3. 编译源代码： mvn compile

4. 编译测试代码：mvn test-compile

5. 运行测试：mvn test

6. 产生 site：mvn site

7. 打包：mvn package

8. 在本地 Repository 中安装 jar：mvn install

9. 清除产生的项目：mvn clean

10. 生成 eclipse 项目：mvn eclipse:eclipse

11. 生成 idea 项目：mvn idea:idea

12. 组合使用 goal 命令，如只打包不测试：mvn -Dtest package

13. 编译测试的内容：mvn test-compile

14. 只打 jar 包: mvn jar:jar

15. 只测试而不编译，也不测试编译：mvn test -skiping compile -skiping test-compile

(-skiping 的灵活运用，当然也可以用于其他组合命令)

16. 清除 eclipse 的一些系统设置:mvn eclipse:clean

ps：

一般使用情况是这样，首先通过 cvs 或 svn 下载代码到本机，然后执行 mvn eclipse:eclipse 生成 eclipse 项目文件，然后导入到 eclipse 就行了；修改代码后执行 mvn compile 或 mvn test 检验，也可以下载 eclipse 的 maven 插件。

mvn -version/-v 显示版本信息

mvn archetype:generate 创建 mvn 项目

mvn archetype:create -DgroupId=com.oreilly -DartifactId=my-app 创建 mvn 项目

mvn package 生成 target 目录，编译、测试代码，生成测试报告，生成 jar/war 文件

mvn jetty:run 运行项目于 jetty 上,

mvn compile 编译

mvn test 编译并测试

mvn clean 清空生成的文件

mvn site 生成项目相关信息的网站

mvn -Dwtpversion=1.0 eclipse:eclipse 生成 Wtp 插件的 Web 项目

mvn -Dwtpversion=1.0 eclipse:clean 清除 Eclipse 项目的配置信息(Web 项目)

mvn eclipse:eclipse 将项目转化为 Eclipse 项目

在应用程序中使用多个存储库

```
<repositories>

<repository>

<id>Ibiblio</id>

<name>Ibiblio</name>

<url>http://www.ibiblio.org/maven/</url>

</repository>

<repository>

<id>PlanetMirror</id>

<name>Planet Mirror</name>

<url>http://public.planetmirror.com/pub/maven/</url>

</repository>

</repositories>
```

```
mvn deploy:deploy-file -DgroupId=com -DartifactId=client -Dversion=0.1.0
-Dpackaging=jar -Dfile=d:\client-0.1.0.jar
-DrepositoryId=maven-repository-inner
-Durl=ftp://xxxxxxx/opt/maven/repository/
```

发布第三方 Jar 到本地库中：

```
mvn install:install-file -DgroupId=com -DartifactId=client -Dversion=0.1.0
-Dpackaging=jar -Dfile=d:\client-0.1.0.jar
```

`-DdownloadSources=true`

`-DdownloadJavadocs=true`

`mvn -e` 显示详细错误 信息.

`mvn validate` 验证工程是否正确，所有需要的资源是否可用。

`mvn test-compile` 编译项目测试代码。 。

`mvn integration-test` 在集成测试可以运行的环境中处理和发布包。

`mvn verify` 运行任何检查，验证包是否有效且达到质量标准。

`mvn generate-sources` 产生应用需要的任何额外的源代码，如 xdoclet。