eclipse+maven+selenium+testNG（reportNG）+git+Jenkins

（**Windows系统**）测试框架（环境）部署。

**2019-3-15**，目前完成的部署是上述的自动化测试环境。

但是Jenkins集成方面还有待完善；selenium是针对web app的浏览器测试，目前的测试对象环境是预先通过其他方式部署好的（放在tomcat中部部署好了），如果war包代码更新后，能够根据测试环境的代码更新后，自动执行回归测试的自动化方案，还没开始尝试。

这个往后继续摸索下。

下面回顾下上述测试框架的搭建方式。

**1.java环境（JDK环境变量）的配置**就不多说了

1）官网下载jdk1.8，打开安装jdk和jre

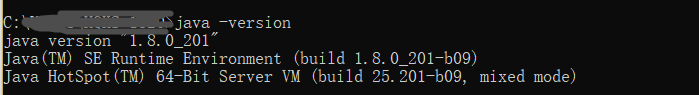
2）然后我的电脑--属性--高级--环境变量--添加系统变量【JAVA\_HOME】，值设为jdk安装目录【C:\Program Files\Java\jdk1.8.0\_151】（而不是jre目录）

3）添加系统变量CLASS\_PATH，值设为【.;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar】

3）修改系统变量path，添加值为【%JAVA\_HOME%\jre\bin】和【%JAVA\_HOME%\bin】



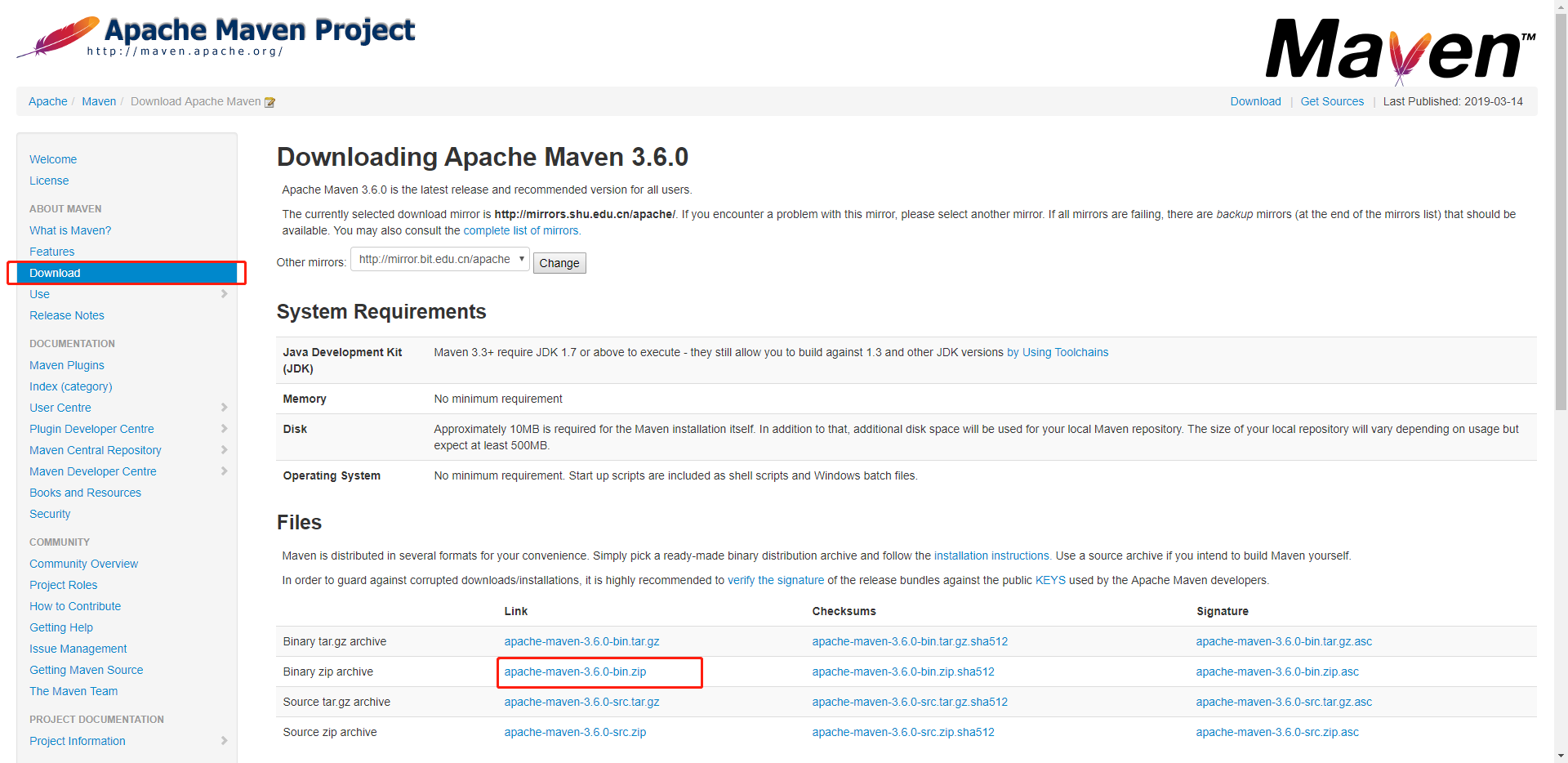
4）配置完后，打开CMD，输入【java -version】，类似下图确认安装完成



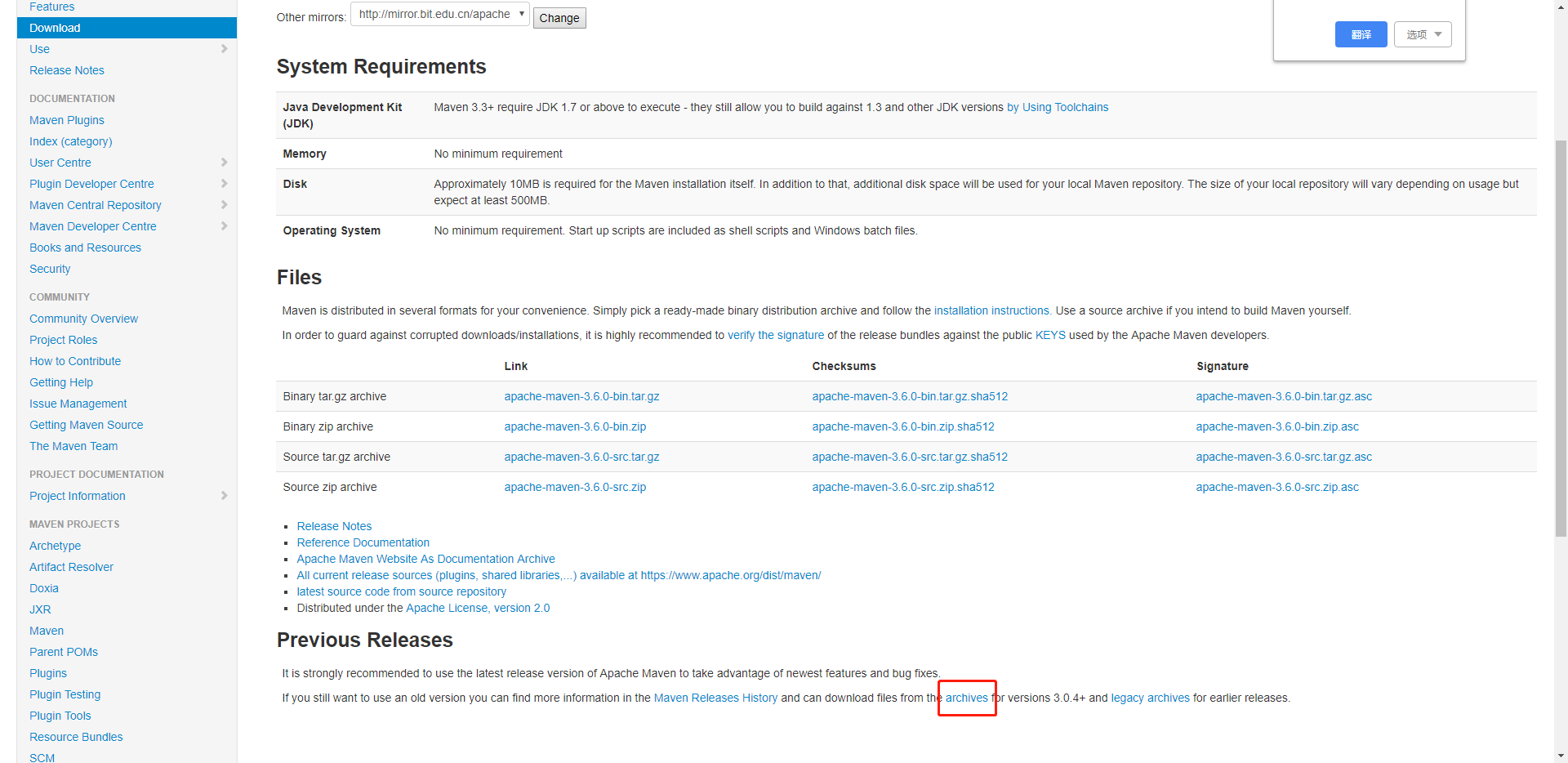
**2.安装maven**

maven有个好处是配置pom.xml环境后，可以自动下载所需要的jar包，不用单独下载导入，此处不多介绍。（也可以使用Ant，使用build.xml的方法。）

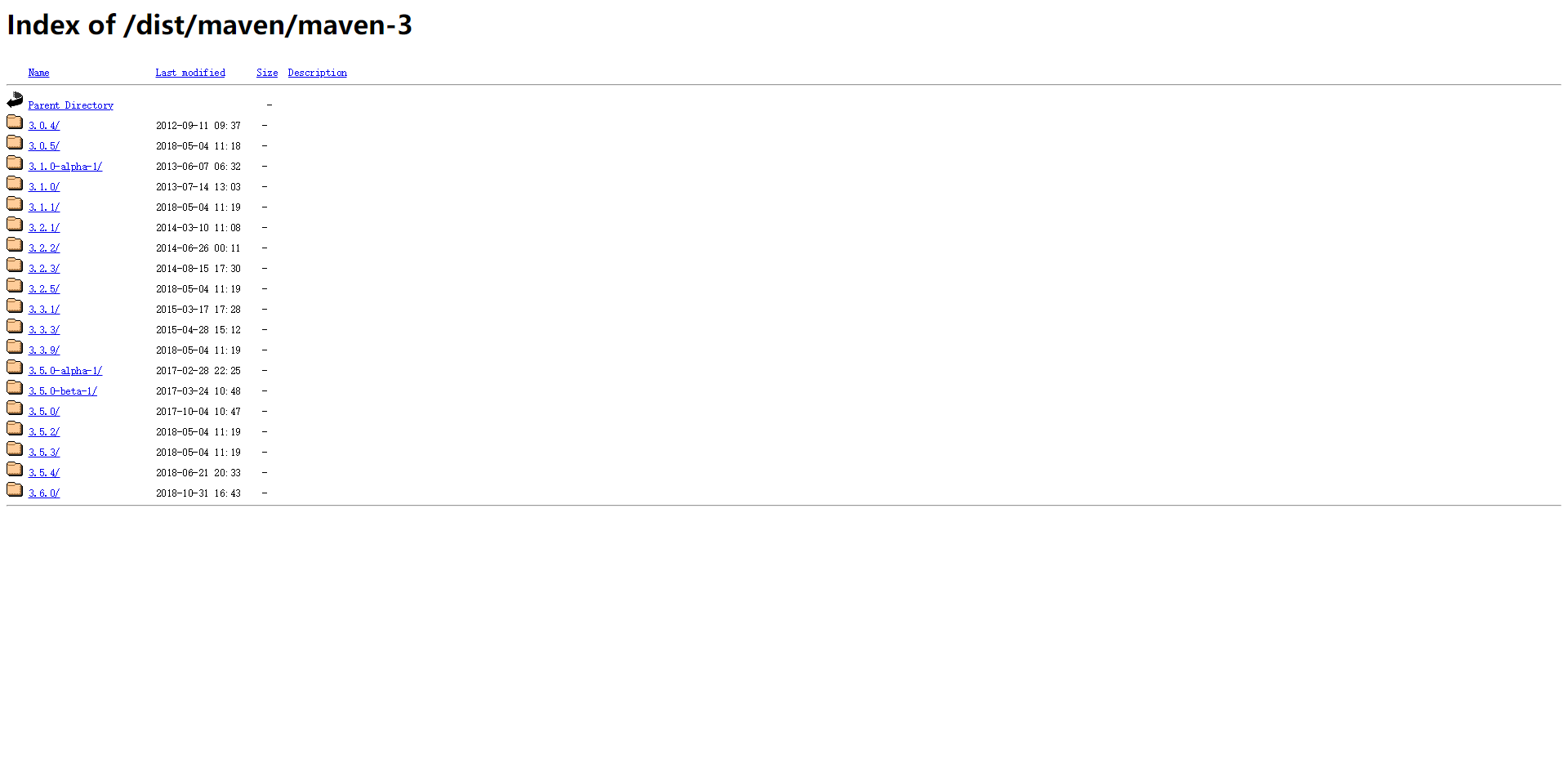
1）在maven官网下载，左侧Download，



想要下载旧版本的话，点击网页下方一个archives的连接，



点击后，如下页面，可以选择想要的版本。



比如目前用的是3.5.0版本，选择binaries二进制zip压缩包。

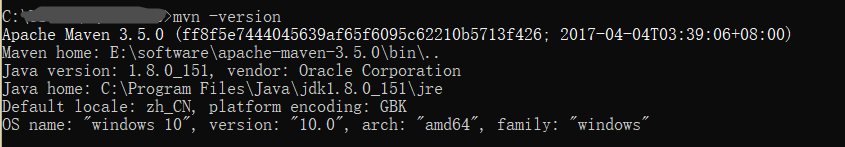


下载后，解压缩得到：



配置环境变量，操作方法类似jdk。添加系统变量【MVN\_HOME】，值填写【E:\software\apache-maven-3.5.0】，系统变量path中添加【%MVN\_HOME%\bin】。

打开CMD，输入【mvn -version】，确认类似下图可以看到maven版本，和java信息，表示安装完成。



2）配置setting.xml文件

这个文件主要配置本地仓库目录以及远程镜像，中央仓库目录。

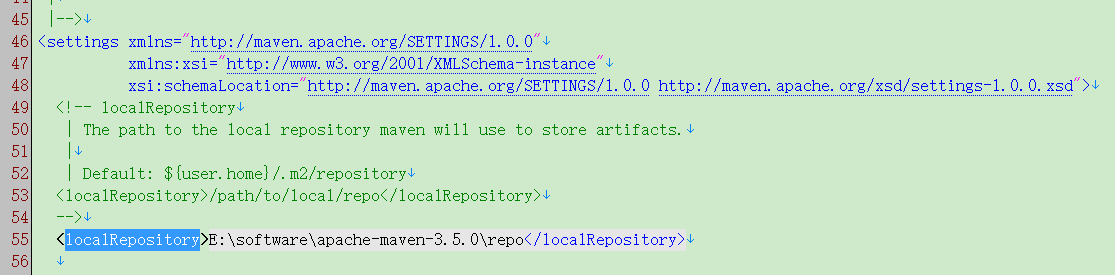
仓库就是引用到的jar的地址。

本地仓库就是需要引用到的jar下载到本地后存放的目录；

远程镜像/中央仓库就是远程服务器的地址。比如阿里云的镜像仓库什么的<http://maven.aliyun.com/nexus/content/groups/public/>

默认setting.xml是在【E:\software\apache-maven-3.5.0\conf】，即maven根目录的conf文件夹下，（1）配置本地仓库

打开后，找到**<localRepository>**，在没有使用<!--\*\*\*-->注释的那一行中间填写本地仓库目录，比如我是在maven根目录下新建的repo，就放在这里面。



（2）配置远程镜像中央仓库

在大约147行左右，在**<mirrors>**标签下配置阿里云仓库如下。

<mirror>

<id>alimaven</id>

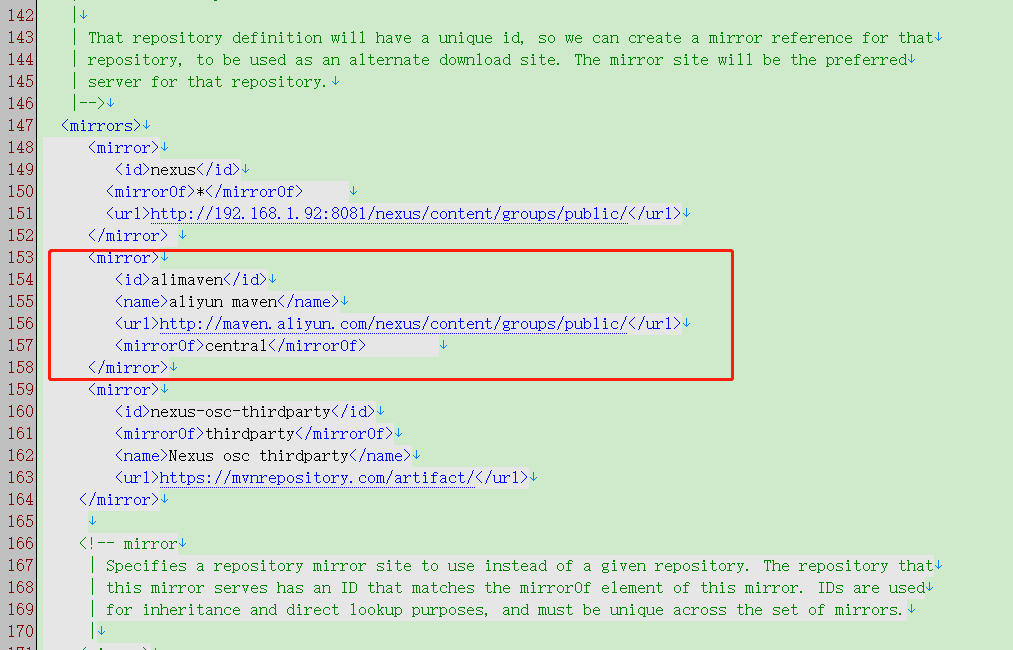
<name>aliyun maven</name>

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

<mirrorOf>central</mirrorOf>

</mirror>

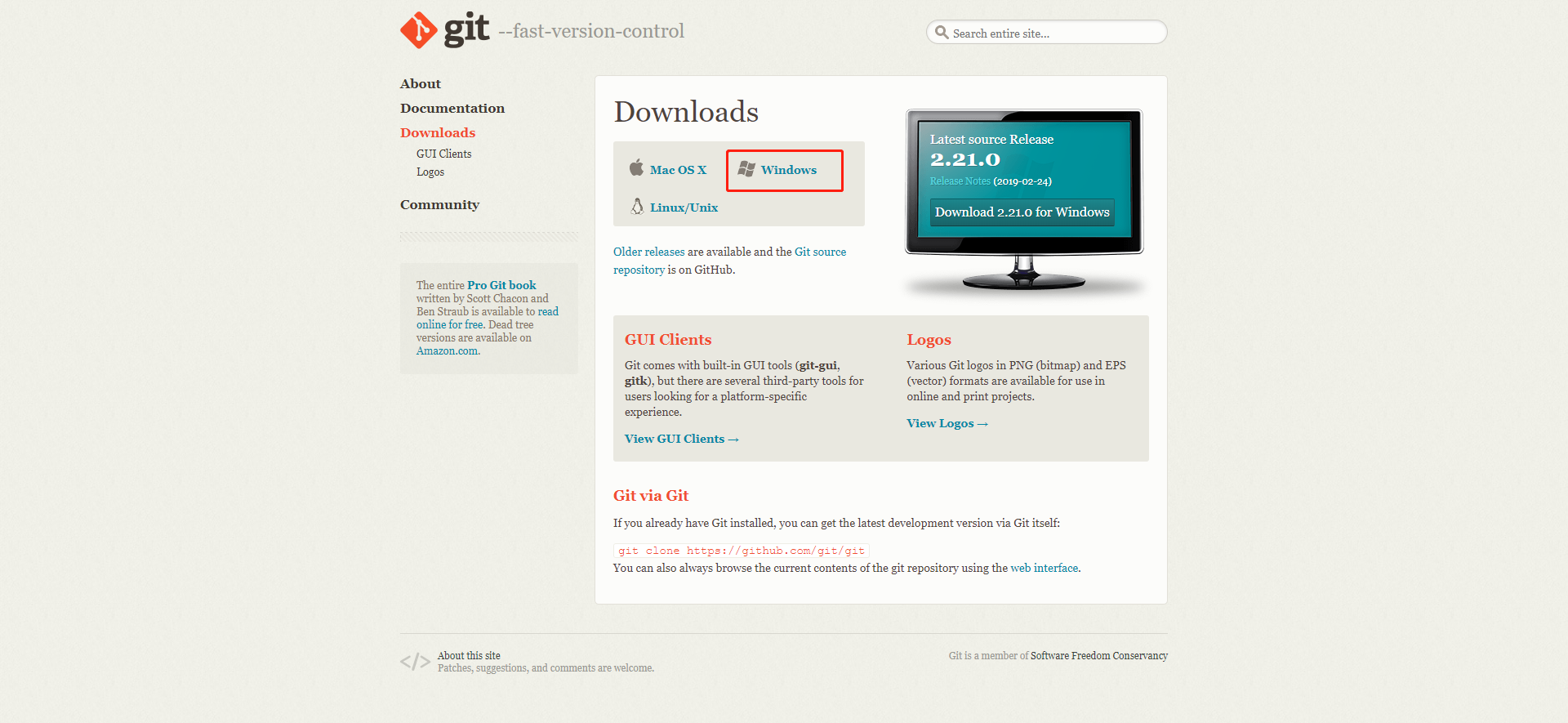
我这里是因为有项目本地仓库，所以添加了好几个仓库。



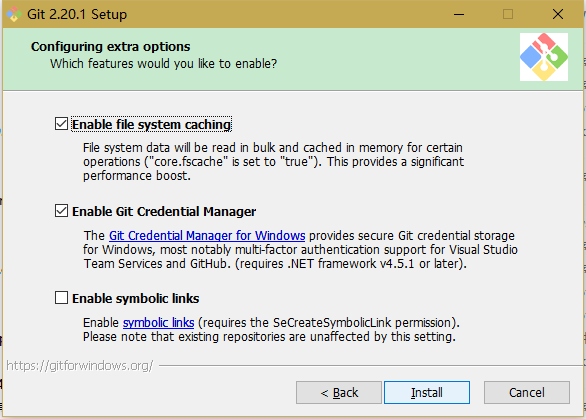
修改后保存即可。（根据实际需要可以配置多个仓库）

**3.Git安装**

1）登录Git官网，点击Windows版，开启下载



下载后，安装，默认一直点击Next即可，然后Install。完成后，会在桌面自动生成Git Bash（和Git GUI两个）快捷方式。（需要.NET 4.5.1环境）

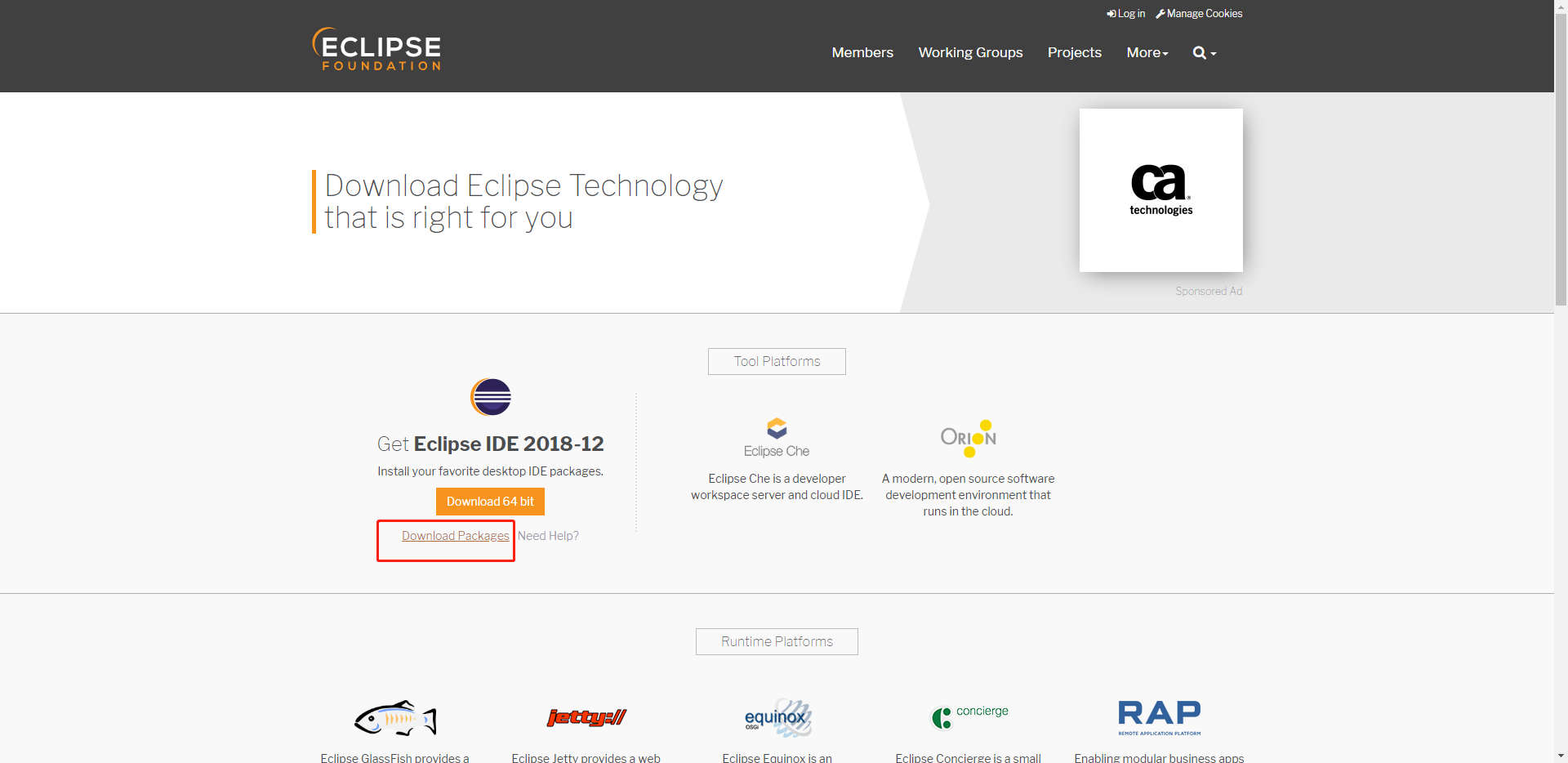


安装完后，可以用CMD输入【git -version】查看一下安装结果，如下图表示安装成功。

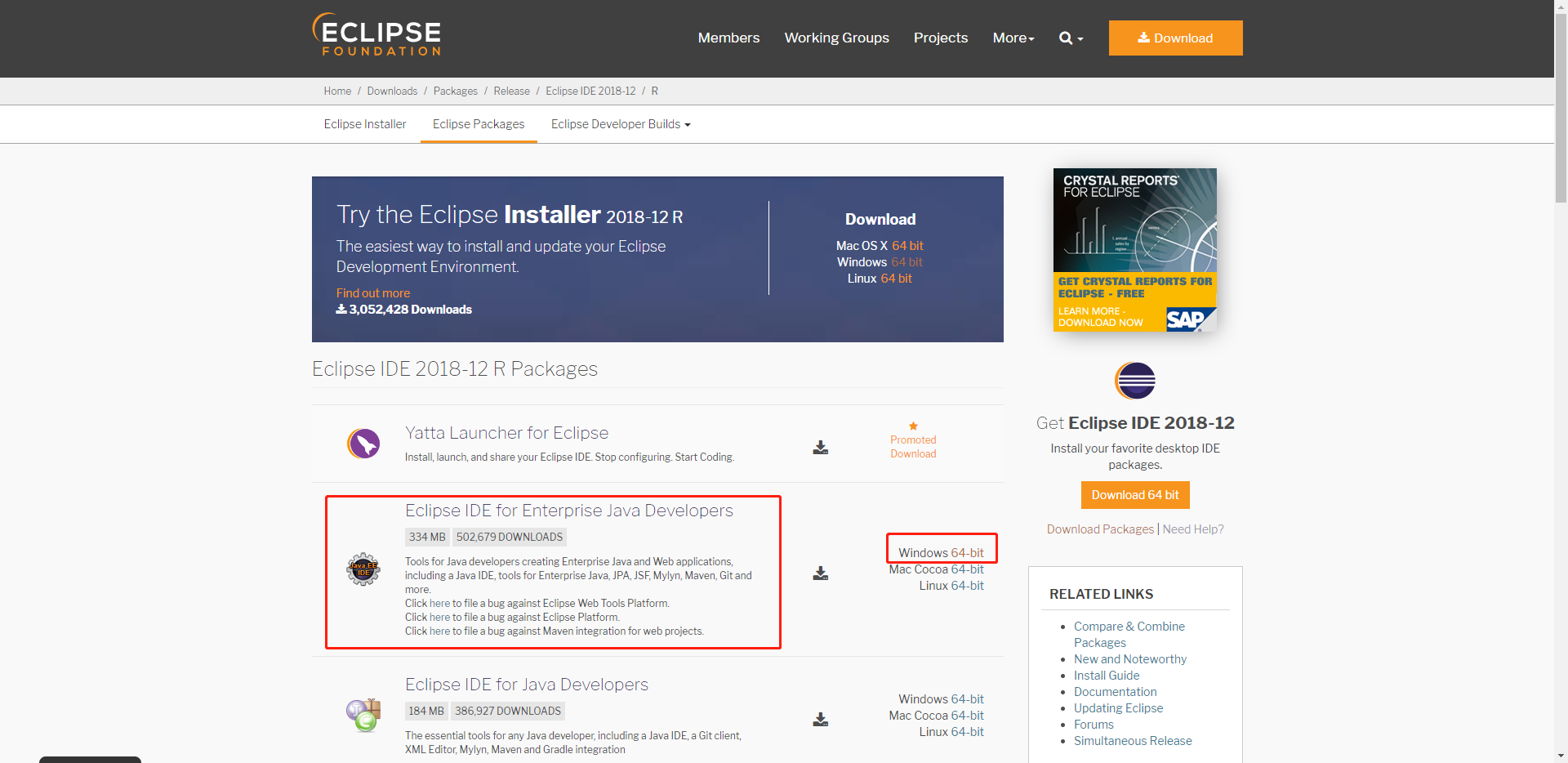
clipboard.png

**4.eclipse安装**

1）进入官网，下载解压缩版，比较方便



然后选择企业java版，Windows-64位



下载后解压缩，

clipboard.png

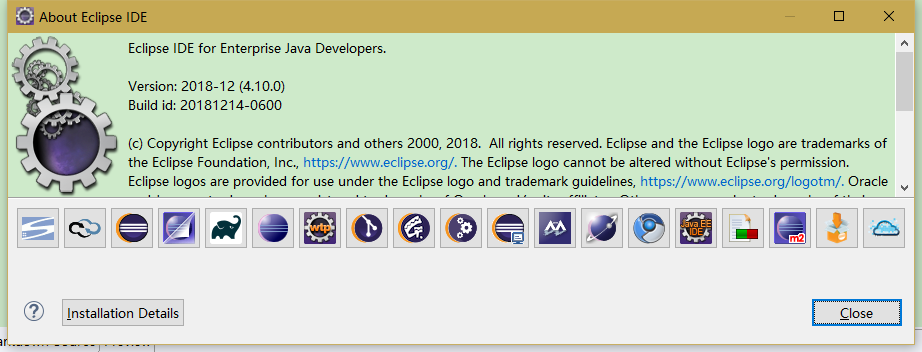
得到



双击eclipse.exe即可打开。（可以右键选择发送到桌面快捷方式方便以后常用，或者添加到任务栏快速启动）关闭引导页面

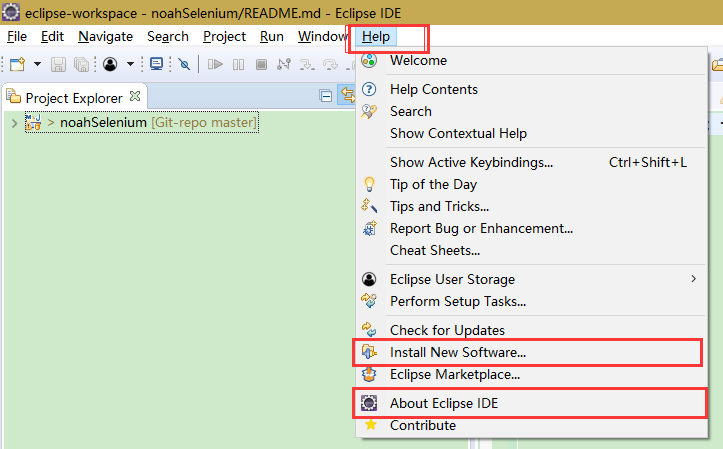
2）安装testNG插件（新版本自带git插件不用额外安装了）。

（下图选择About Eclipse IDE可以查看版本信息）

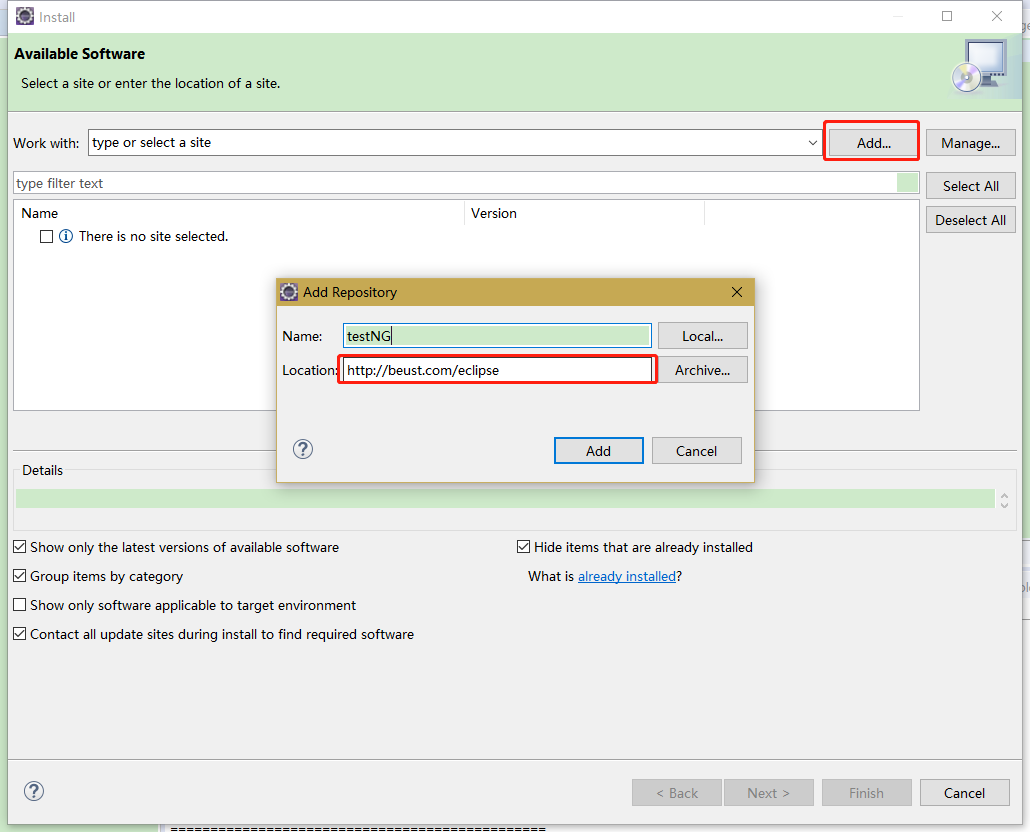


安装testNG步骤：

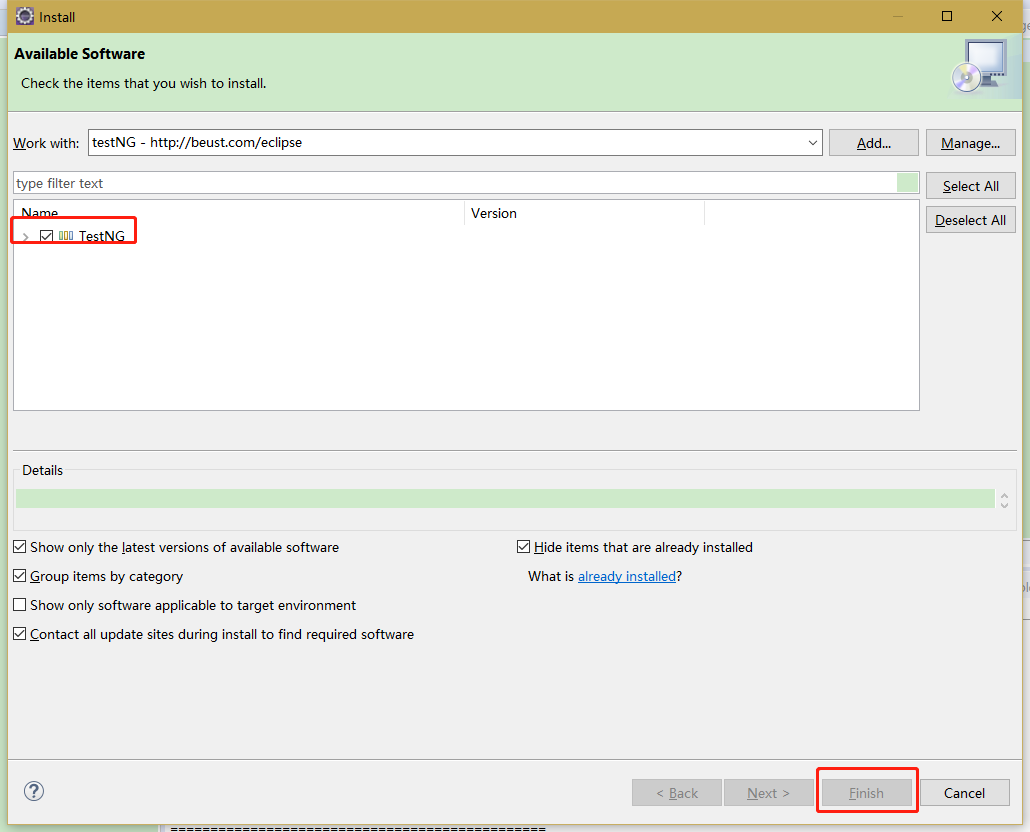
菜单栏的Help--Install New Software，



在打开的弹框中，点击Add，Location输入：【<http://beust.com/eclipse>】，Name随意，填写testNG即可。



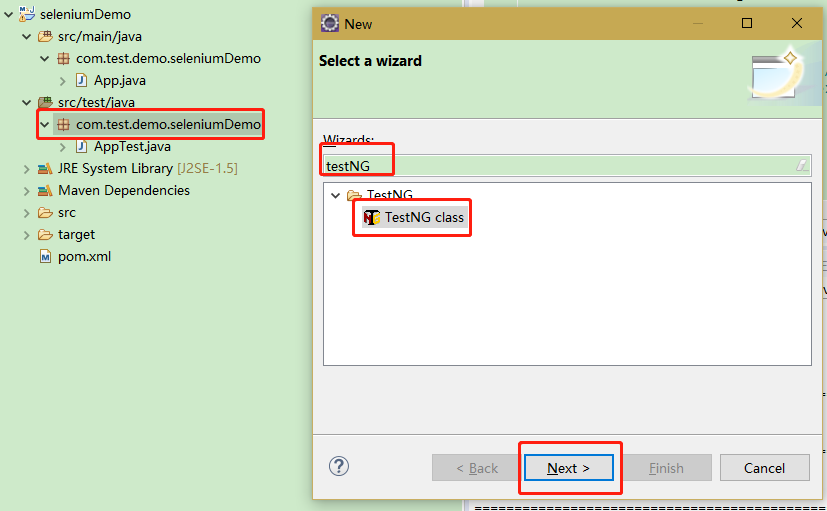
点击Add按钮，过会儿会找到插件，勾选，点击Finish或者Next（此处已经安装过所以无法点击）



后面页面选择同意许可，等待安装后，根据提示重启Eclipse。

验证安装结果：

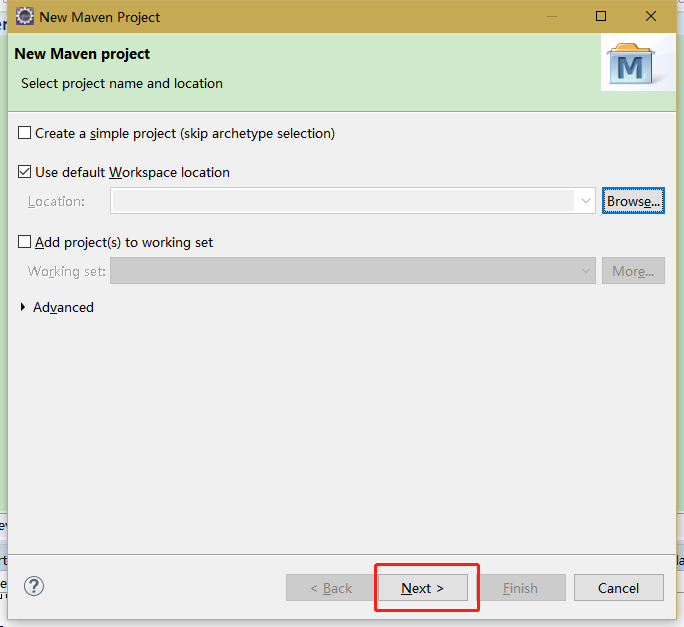
对着项目中的test下的包目录右击鼠标，选择New--Other，打开窗口中输入testNG，看到有查询到testNG，表示安装成功。



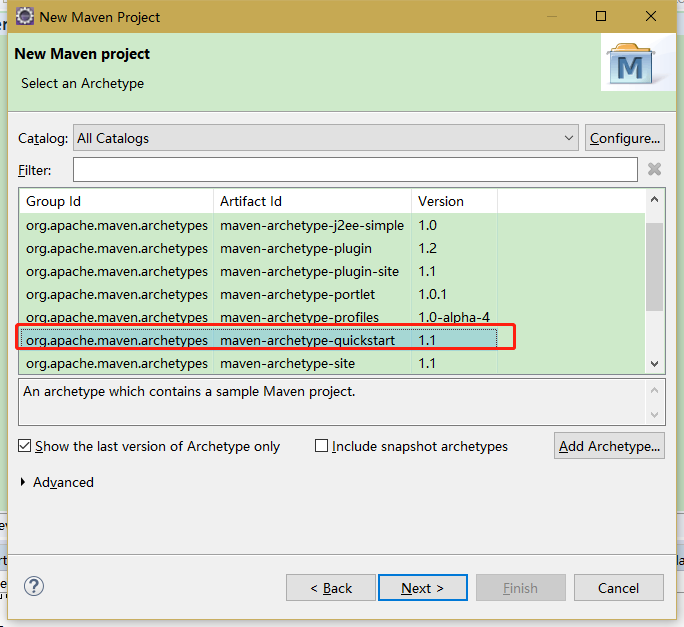
3）新建maven项目

点击菜单栏File--New--Maven Project。（新版本可以直接选择）

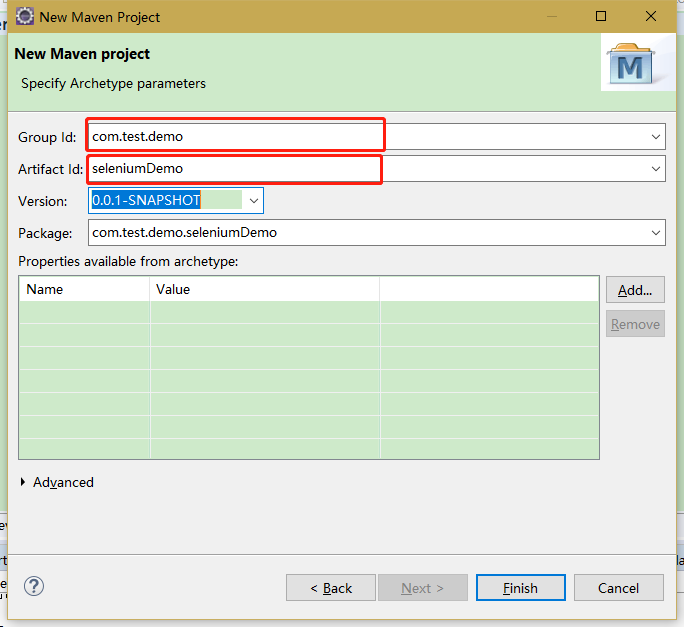
下图提示是否使用默认工作空间（就是项目文件存放的位置，根据需要自行修改），可以直接Next



选择maven-archetype-quickstart，继续Next



填写Group Id和Artifact Id。Group Id就是package，也是文件目录结构名，一般是com.xxx.xxx，Artifact Id是项目名称。填写完，点击Finish

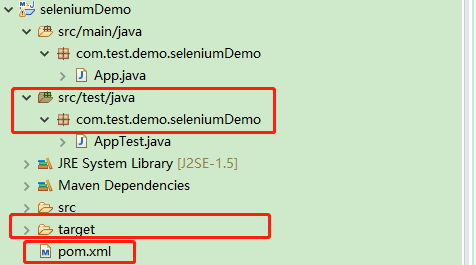


然后生成项目如下，名称就是seleniumDemo，可以看到src是代码目录，主要分为main和test，一般main下面放源代码，test下面的源代码的测试代码。

（1）这里不多说明，主要是在test下面添加测试代码。

（2）target是代码编译生成的文件和打包后存放的默认路径

（3）pom.xml是maven项目的一些配置，主要是基本信息，还有项目所引用的外部jar在这个文件中作为依赖dependency导入。



生成的默认pom.xml是这样：



因为使用selenium+testNG转reportNG，所以需要引用selenium，testNG和reportNG的依赖包，下面直接在pom.xml中尝试添加依赖，主要用到selenium-java，testNG，guice，velocity-dep，log4j，reportng等jar包。其他一些重要配置也标注了红色。

配置完后的pom.xml如下：

————————————被遗忘的分割线————————————

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.tbl.test</groupId>

<artifactId>noahSelenium</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<name>noahSelenium</name>

<url>http://maven.apache.org</url>

<!-- <properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties> -->

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<argLine>-Dfile.encoding=UTF-8</argLine>

<!-- <maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target> -->

<xmlFileName>testNG.xml</xmlFileName>

</properties>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.14.0</version>

<scope>compile</scope>

<!--

scope标签中对应值的解释：

\* compile，缺省值，适用于所有阶段，会随着项目一起发布。

\* provided，类似 compile，期望 JDK、容器或使用者会提供这个依赖。如 servlet.jar。

\* runtime，只在运行时使用，如 JDBC 驱动，适用运行和测试阶段。

\* test，只在测试时使用，用于编译和运行测试代码。不会随项目发布。

\* system，类似 provided，需要显式提供包含依赖的 jar， Maven 不会在 Repository 中查找它。

-->

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.testng/testng -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.11</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/com.google.inject/guice -->

<dependency>

<groupId>com.google.inject</groupId>

<artifactId>guice</artifactId>

<version>4.1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/velocity/velocity-dep -->

<dependency>

<groupId>velocity</groupId>

<artifactId>velocity-dep</artifactId>

<version>1.4</version>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>org.uncommons</groupId>

<artifactId>reportng</artifactId>

<version>1.1.4</version>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<build>

<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

<plugins>

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

<compilerArgs>

<arg>-Xlint:unchecked</arg>

<arg>-Xlint:deprecation </arg>

<!--<arg>endorseddirs=${endorsed.dir}</arg>-->

</compilerArgs>

</configuration>

</plugin>

<!-- 添加插件 关联testNg.xml -->

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<version>2.17</version>

<configuration>

<testFailureIgnore>true</testFailureIgnore>

<suiteXmlFiles>

<suiteXmlFile>**testNG.xml**</suiteXmlFile>

</suiteXmlFiles>

<properties>

<!--刚开始这里没有设置默认监听false，所以没有启用reportNG功能。-->

<property>

<name>usedefaultlisteners</name>

<value>**false**</value>

</property>

<!--Setting ReportNG listener-->

<property>

<name>listener</name>

<value>**org.uncommons.reportng.HTMLReporter, org.uncommons.reportng.JUnitXMLReporter**</value>

</property>

</properties>

<workingDirectory>**target**/</workingDirectory>

<forkMode>always</forkMode>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

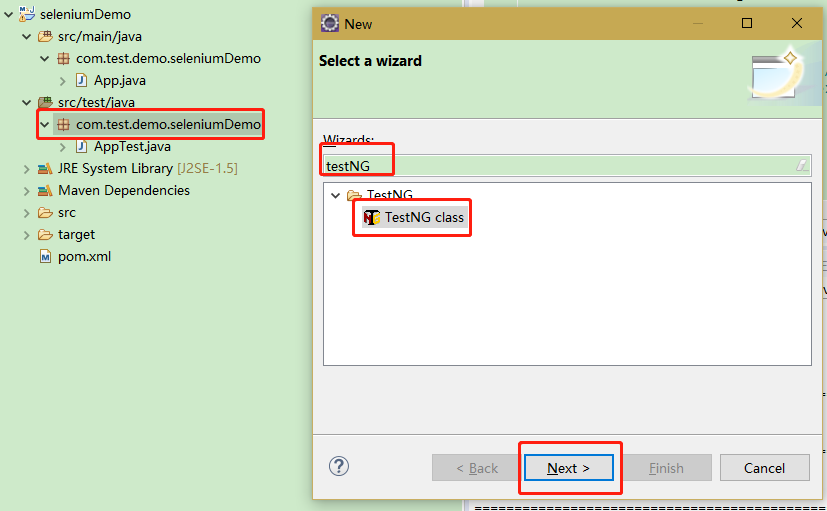
</project>

————————————被遗忘的分割线————————————

配置完pom.xml后，会自动下载相关jar包。（还是右下角会提示选择auto什么的，自动更新，貌似这是IDEA的选项？记不清了）

4）创建第一个测试代码文件

对着项目中的test下的包目录右击鼠标，选择New--Other，打开窗口中输入testNG，选择TestNG class，然后点击Next

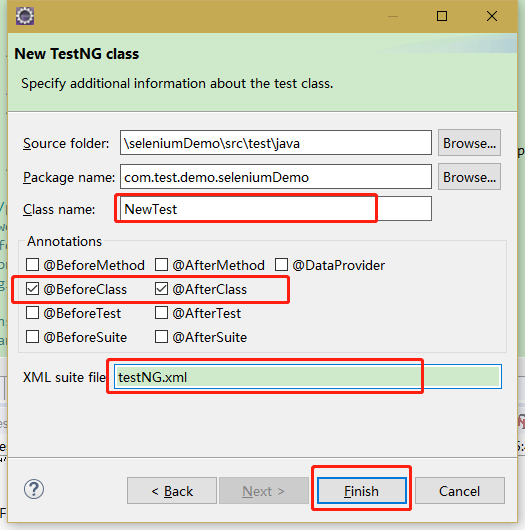


输入类名Class Name，可以选择勾选@BeforeClass和@AfterClass组合。

表示在测试类执行前预先执行标记有@BeforeClass的方法，测试类执行后再执行标记有@AfterClass的方法；或者选择@BeforeMethod和@AfterMethod组合，表示每个测试方法执行前就执行一遍标记有@BeforeMethod的方法，@AfterMethod同理。

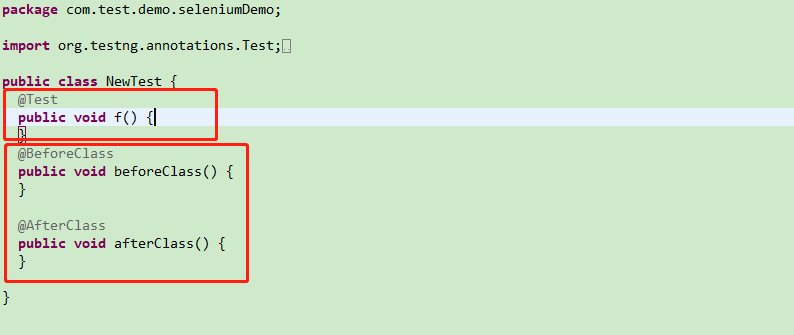
具体各选项含义可以自行百度。

XML suite file输入testNG.xml，点击Finish。

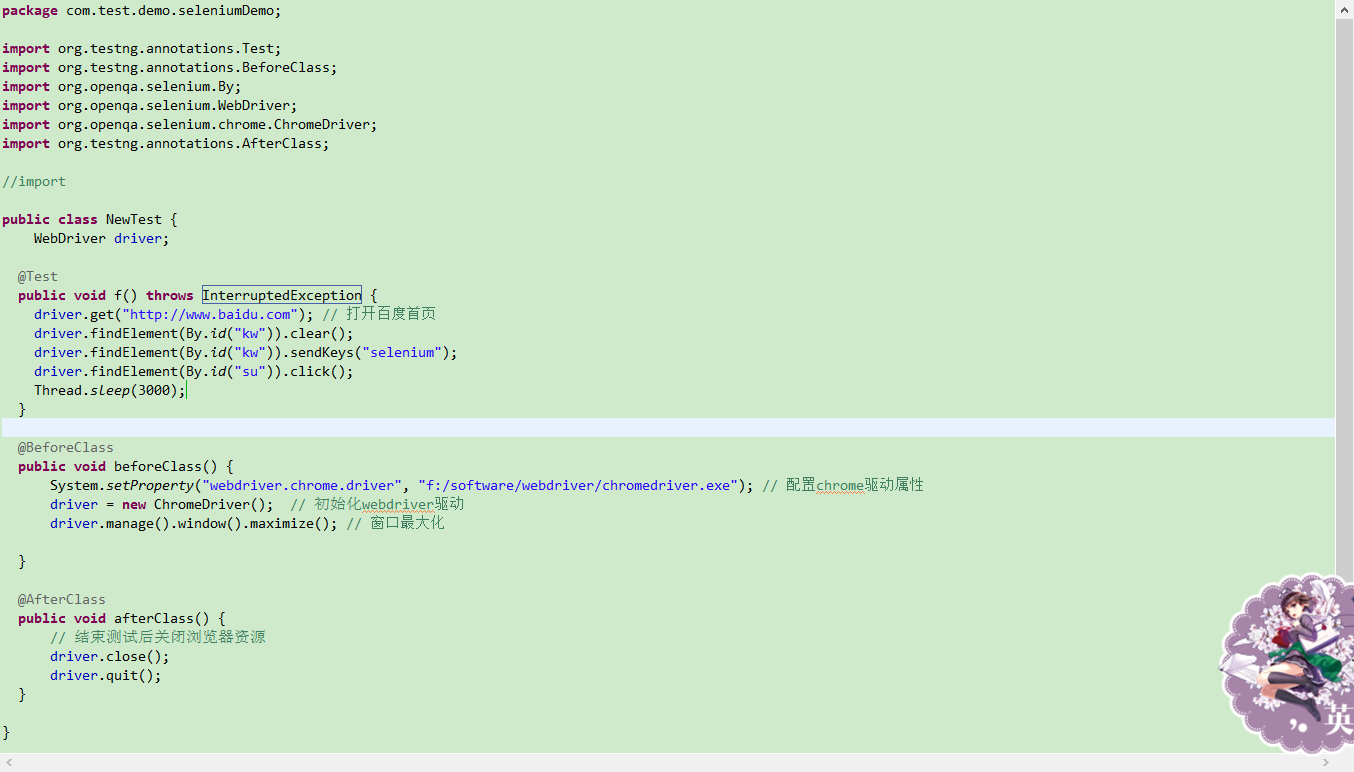


然后生成测试类NewTest：标记有@Test的方法是测试的本体，测试内容写在这个方法内。

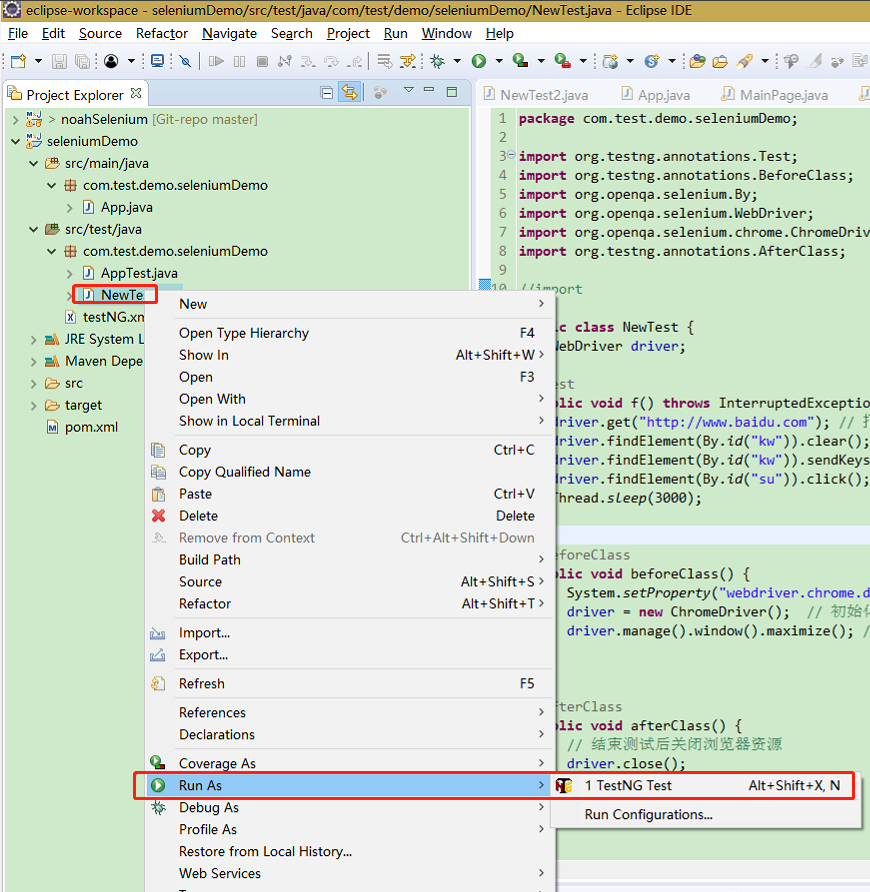
@BeforeClass和@AfterClass就是上面介绍过的也自动生成了。



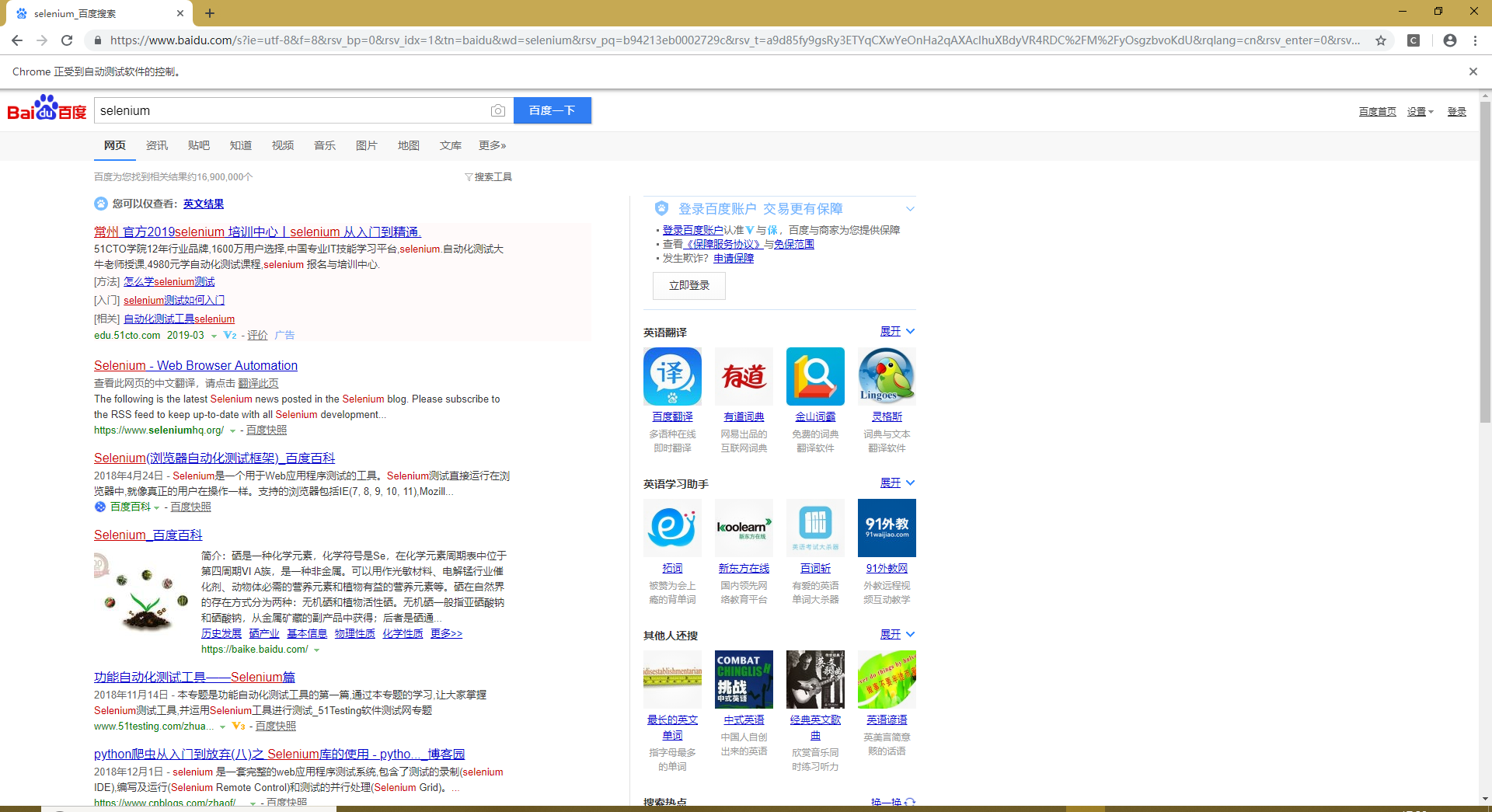
写一个简单的selenium测试类。打开百度首页，搜索selenium。



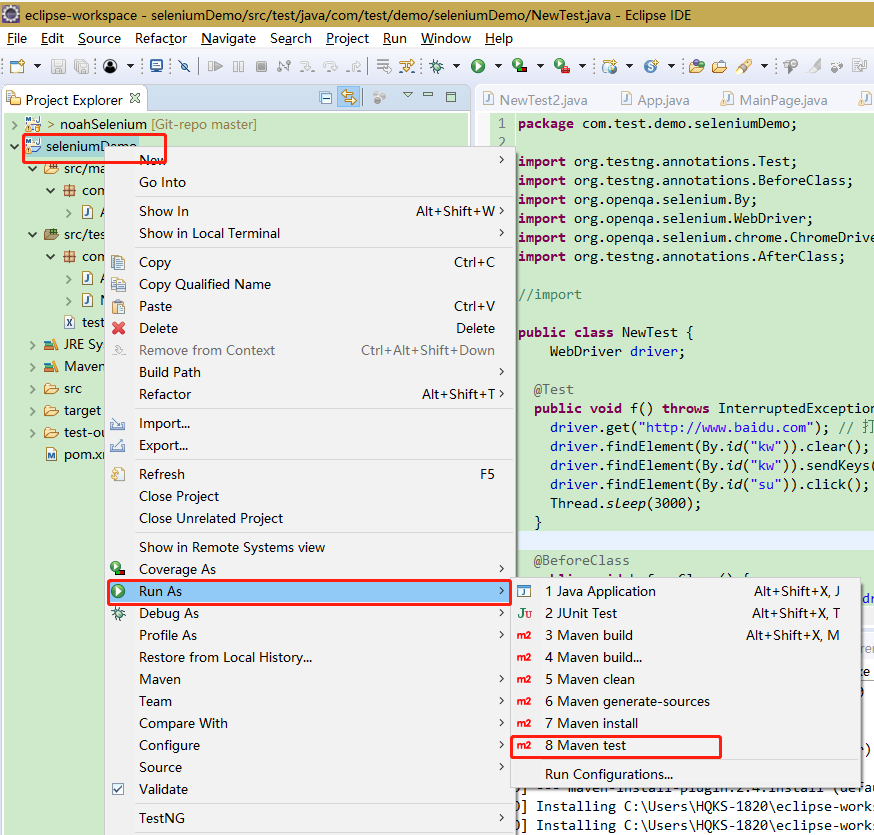
如下图，选中测试类，右击，选择Run As，选择TestNG Test，



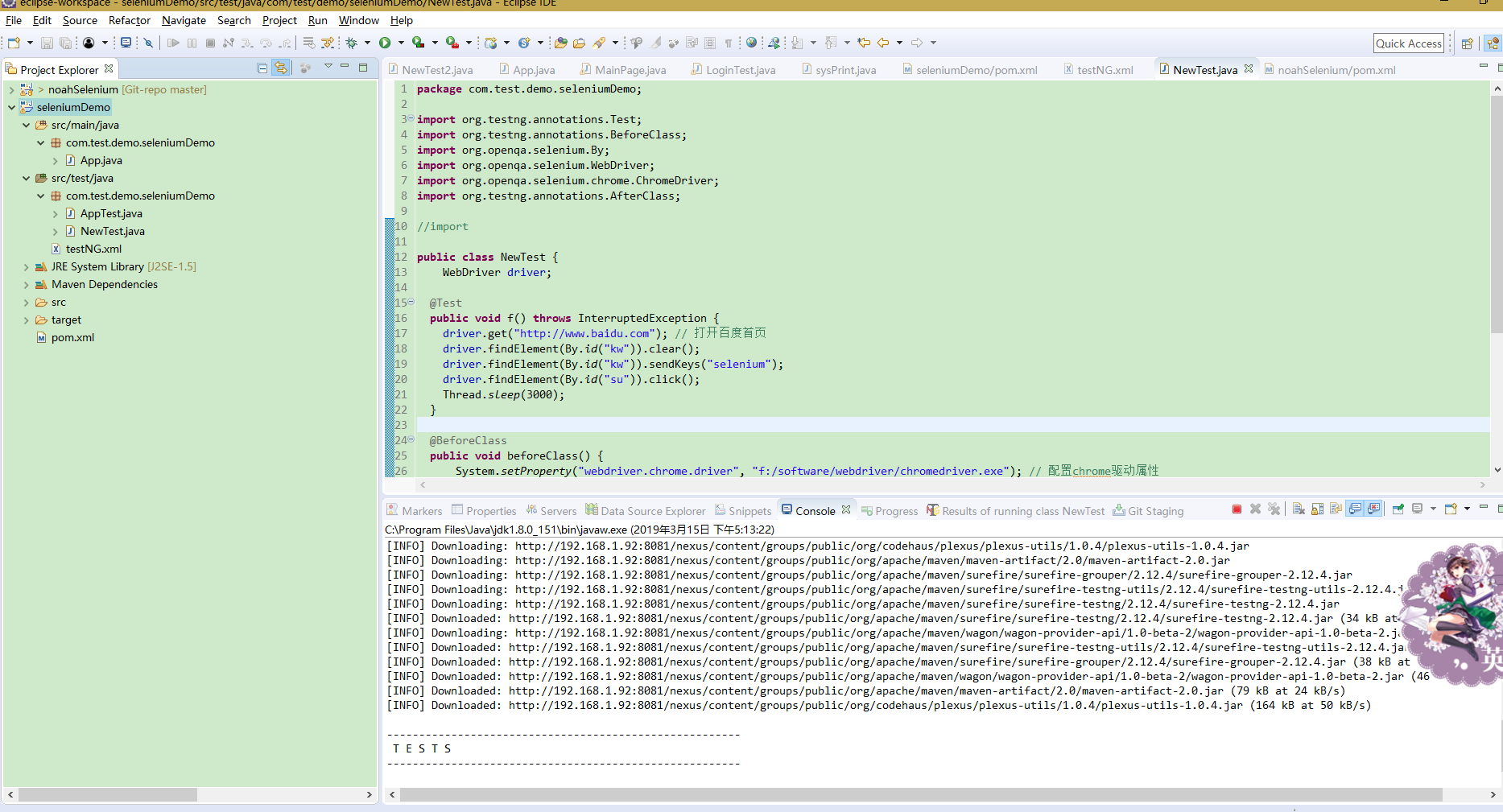
然后开始执行测试类，可以看到自动打开chrome浏览器，百度搜索selenium。



也可以选中项目，右击Run As --Maven test，

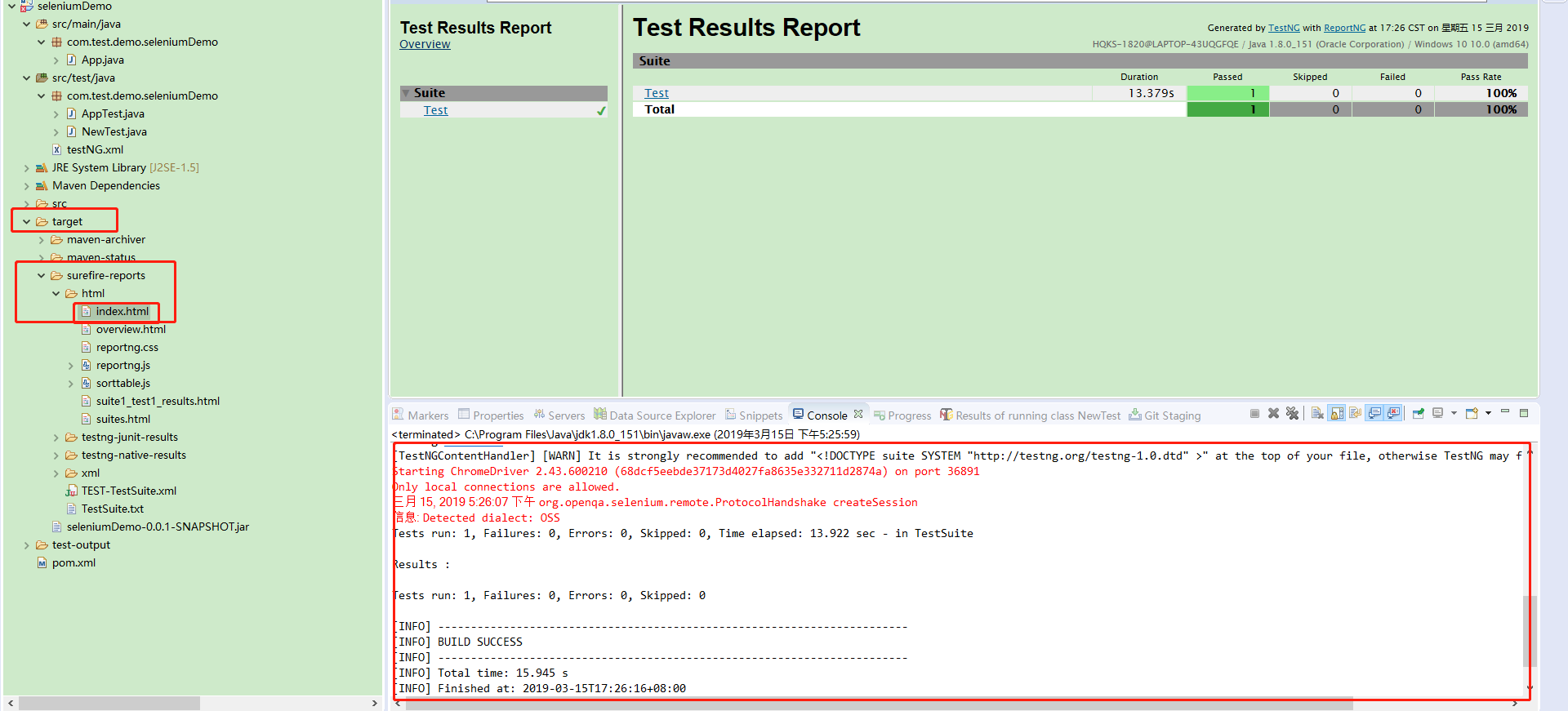


首次运行maven会自动下载一些必备的包。



然后也会自动打开浏览器，百度搜索selenium。

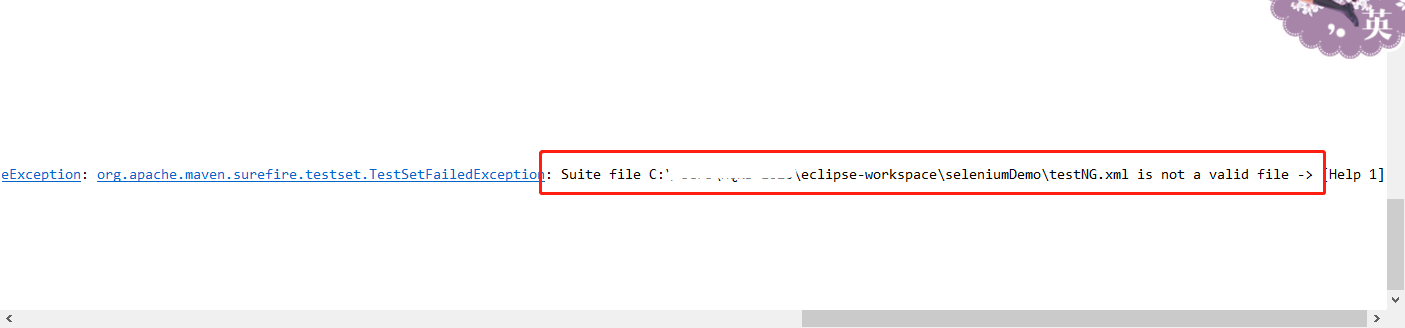
可以看到控制台提示build success。然后打开项目目录下的【target\surefire-reports\html\index.html】，可以看到reportNG（testNG的优化版）报告。



——————————注意点开始——————————

**【注意】**

1）.中途可能会报错：



表示testNG.xml找不到，查看现在testNG.xml是在src/test/java目录下，他是去根目录找的，因为我们配置的pom.xml中是在根目录：



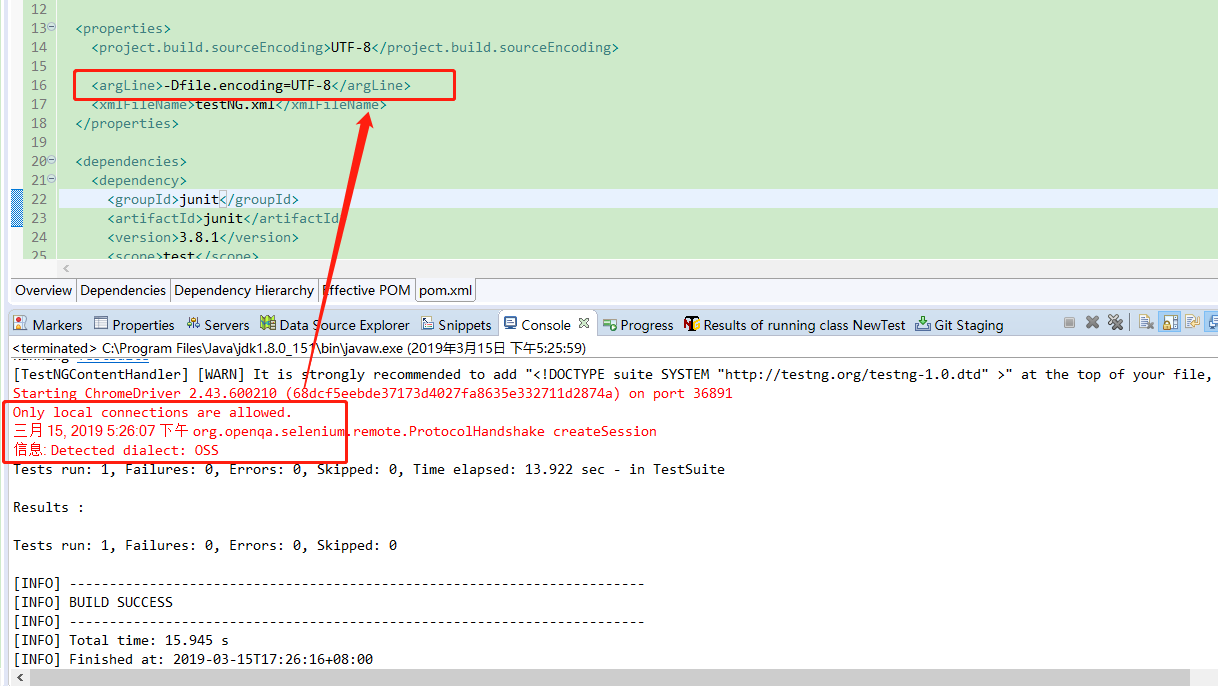
则我们把testNG.xml位置更改成



然后再次运行Run as-- Maven test，可以成功执行。

2）.如果控制台中文显示乱码，可以再pom.xml里**<properties>**标签下加上：

<argLine>-Dfile.encoding=UTF-8</argLine>



3）.测试报告乱码

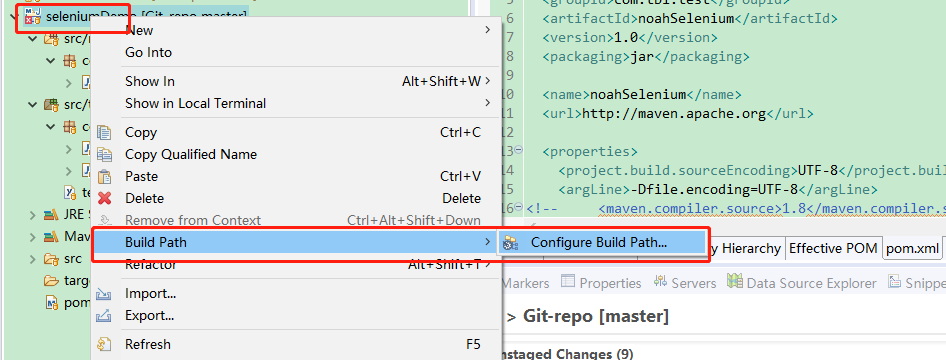
如果下图中中文显示乱码，可以下载下面的版本，替换掉本地仓库目录中maven自动下载下来的org\uncommons\reportng\1.1.4\reportng-1.1.4.jar

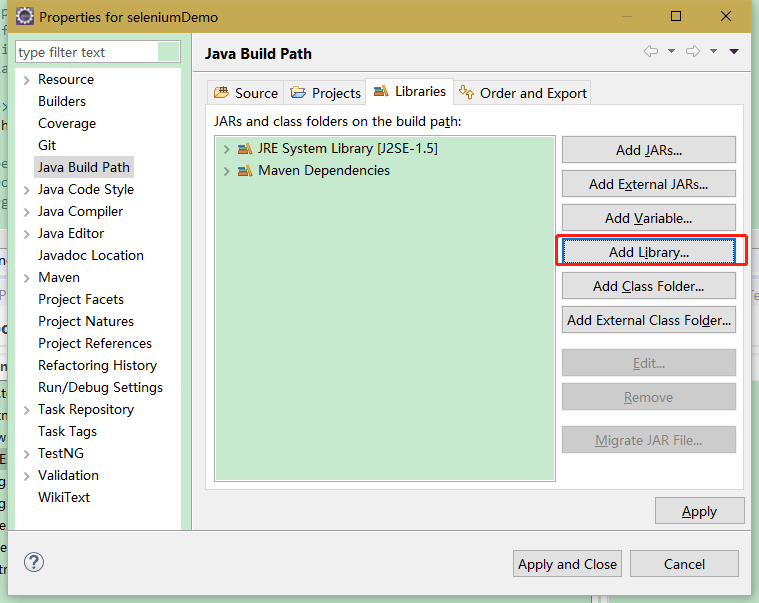
链接：https://pan.baidu.com/s/1pd78z4AKhdIuqLW4UhURng

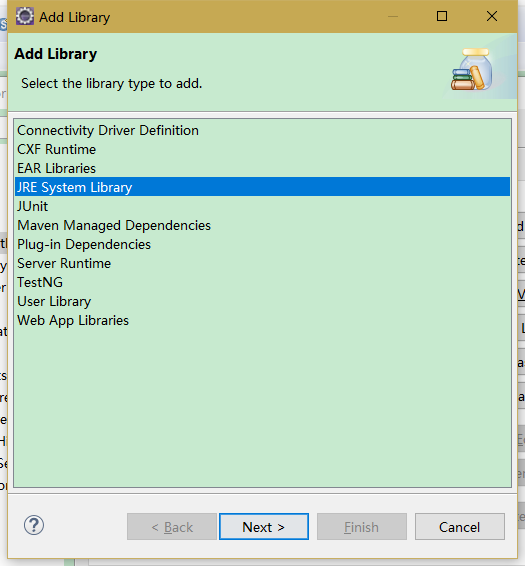
提取码：ajh8



4）期间利用maven运行时遇到的有问题有找不到类文件，需要配置下build path.

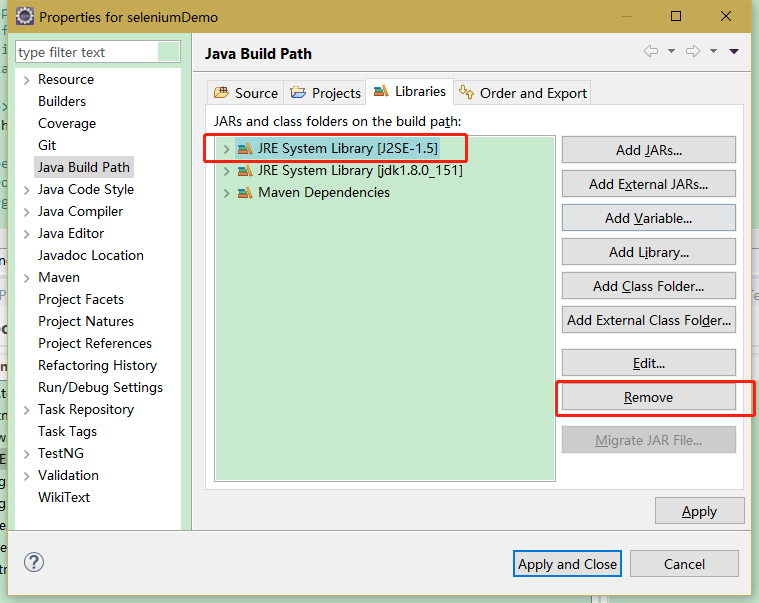








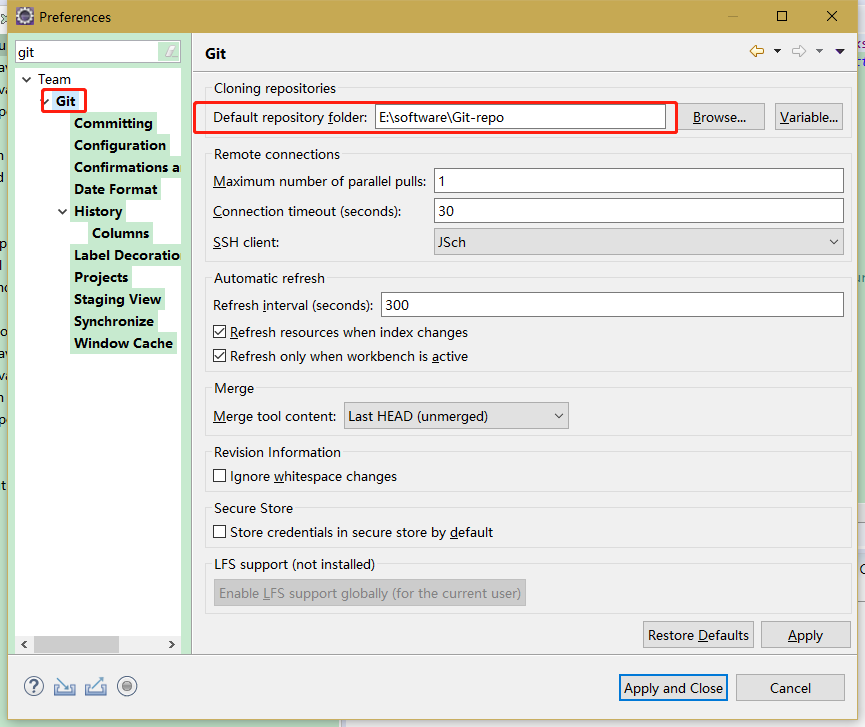
点击确定。删除掉默认的1.5版本。



——————————注意点结束——————————

5）eclipse配置Git

（1）菜单栏Window--Preferences--输入Git，查询得到的结果中选择Git，可以选择本地Git的本地仓库目录

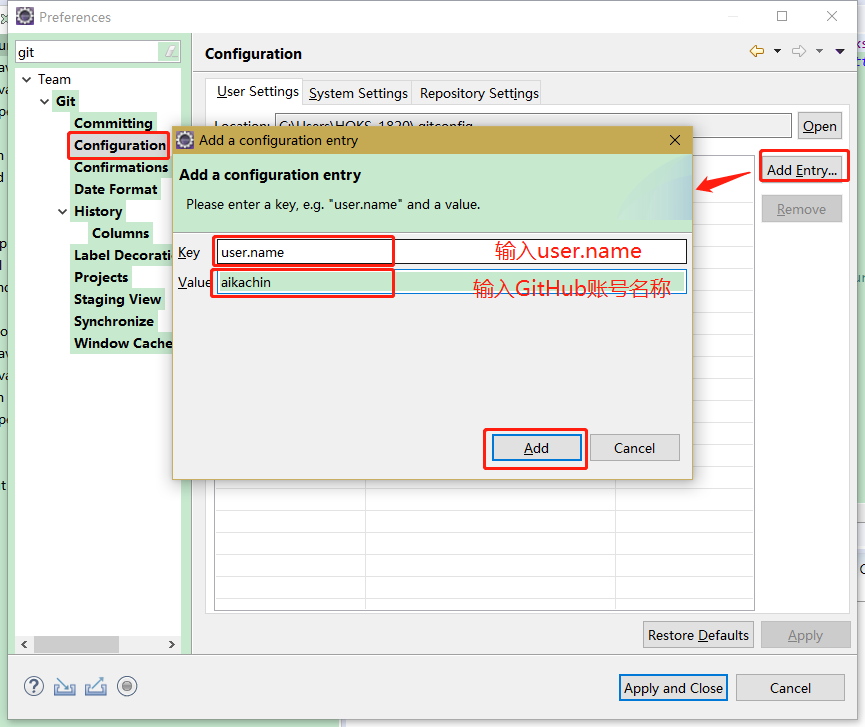


（2）添加Git账号配置

选择Configuration，点击Add Entry，添加Git账号配置。

添加2次，第一次是账号名称，Key：user.name，Value输入自己账号名称;

第二次是账号邮箱。Key：user.email，Value输入自己账号邮箱。



添加完后，如图



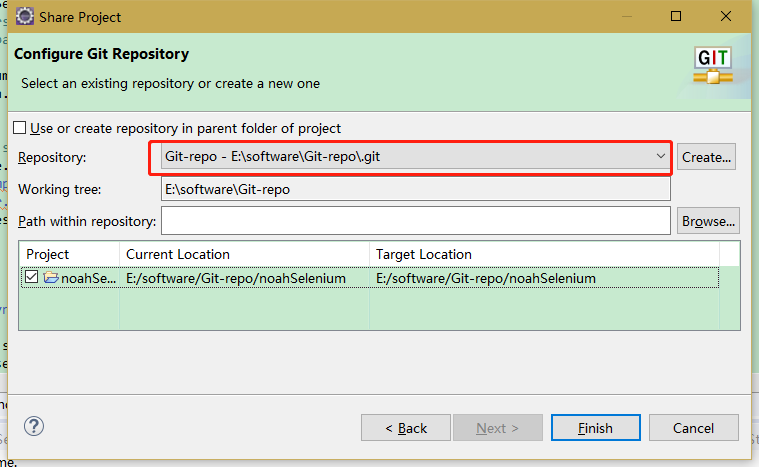
点击确定完成配置。

3）上传项目直Git

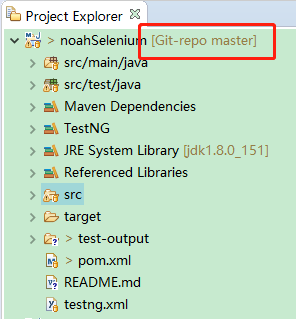
选择项目，右击Team--Share Project，默认跳出Git页面，或者选择Git（安装了SVN插件的话）



选择项目地址，勾选项目，点击Finish。

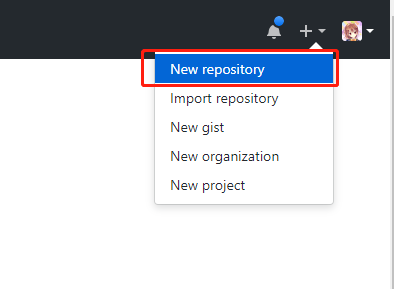


完成项目跟Git的关联后，项目旁显示，表示关联成功：

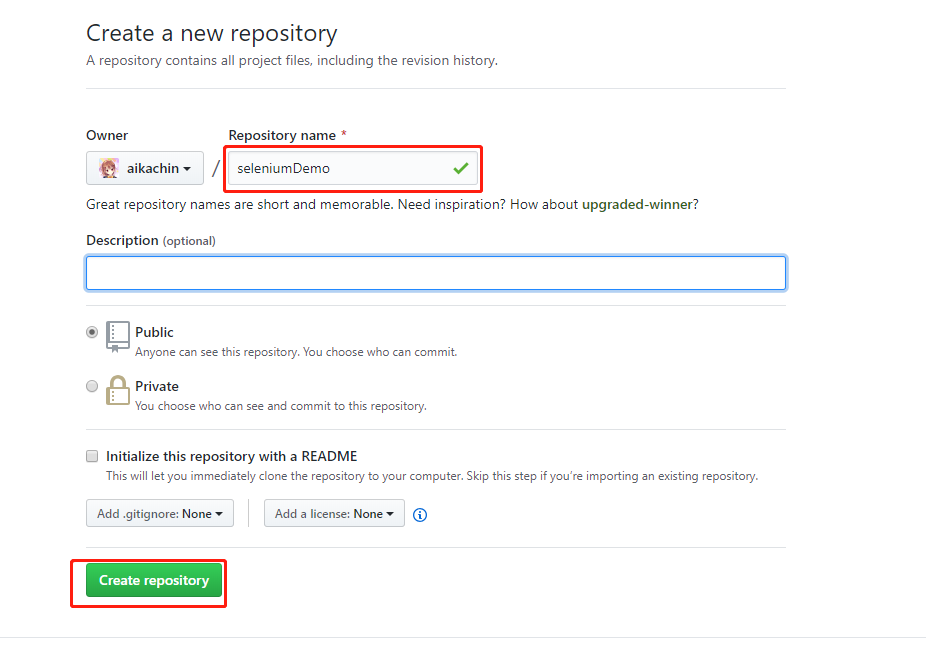


4）登录GitHub，创建一个新项目用来上传自己的项目测试代码。

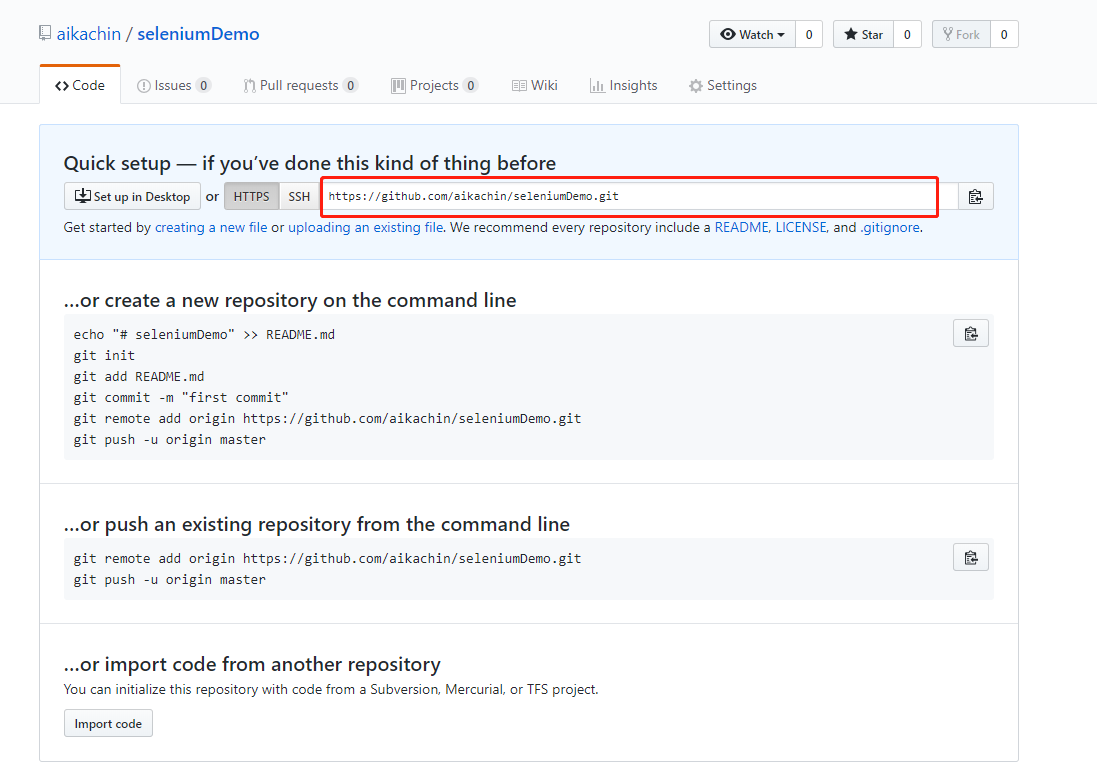
登录GitHub，右上角新建



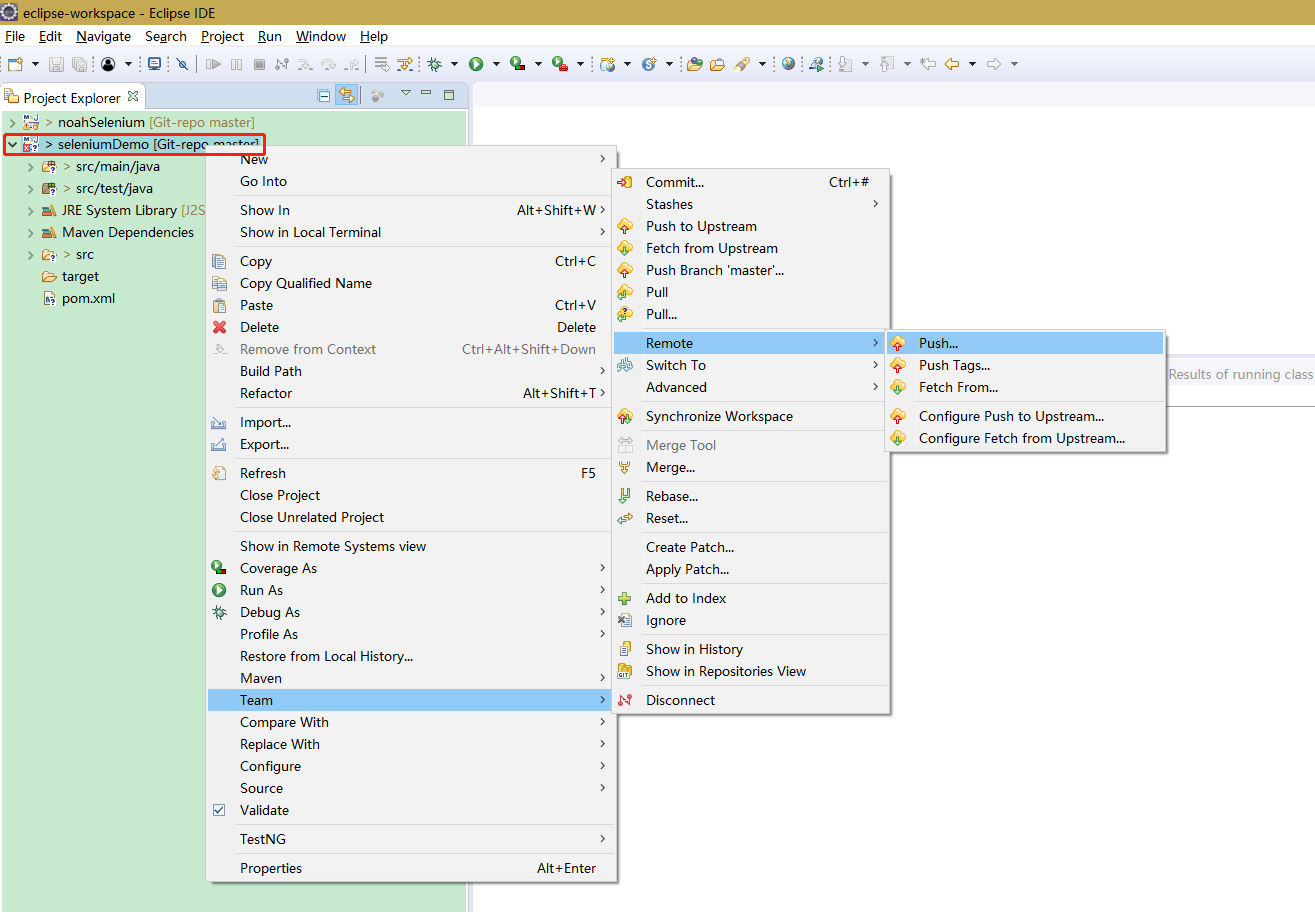
然后，配置新项目，输入名称，默认Public，点击Create repository即可



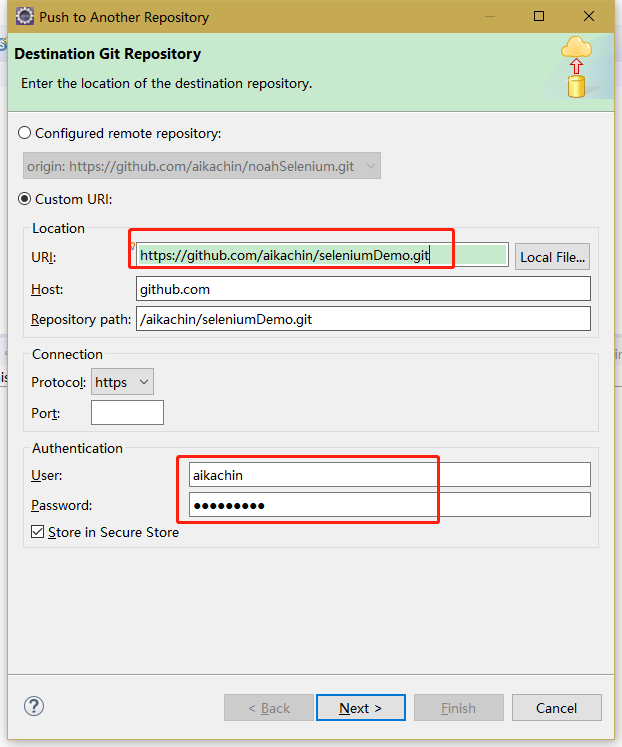
然后进入这个页面，复制项目地址，在eclipse中上传到Git时会有用到。



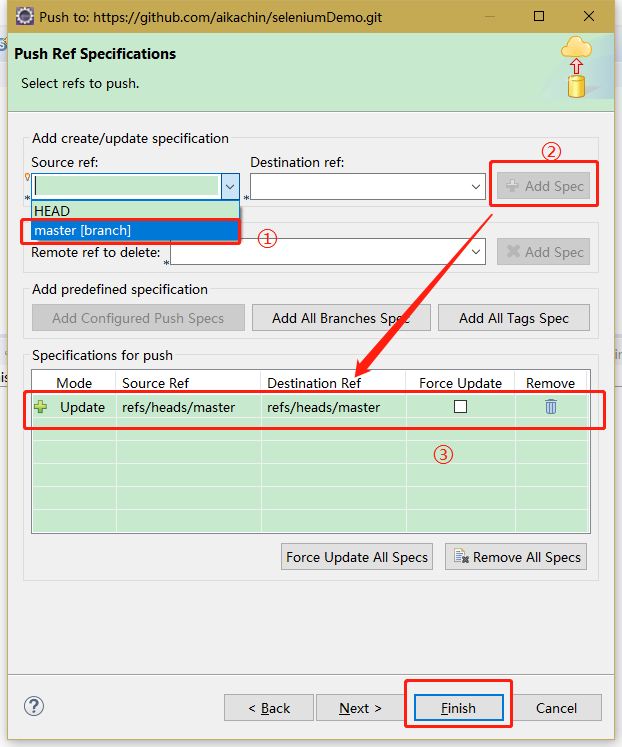
返回eclipse，选择项目，右击，按下图进行选择Team，Push，



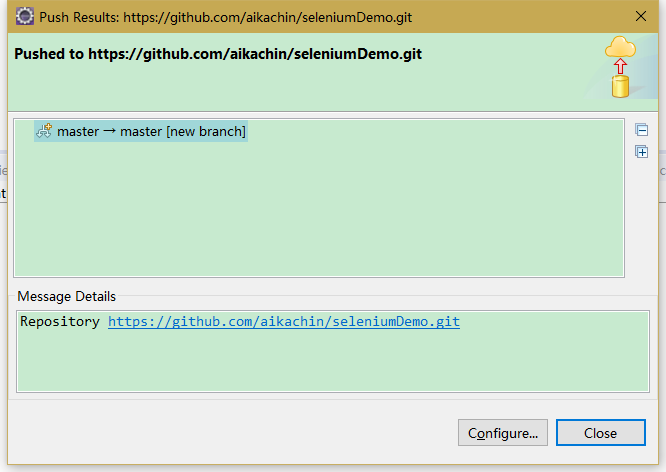
在打开的页面中输入上面复制的Git上的项目地址，以及号名和密码。



点击Next，然后按下图操作：

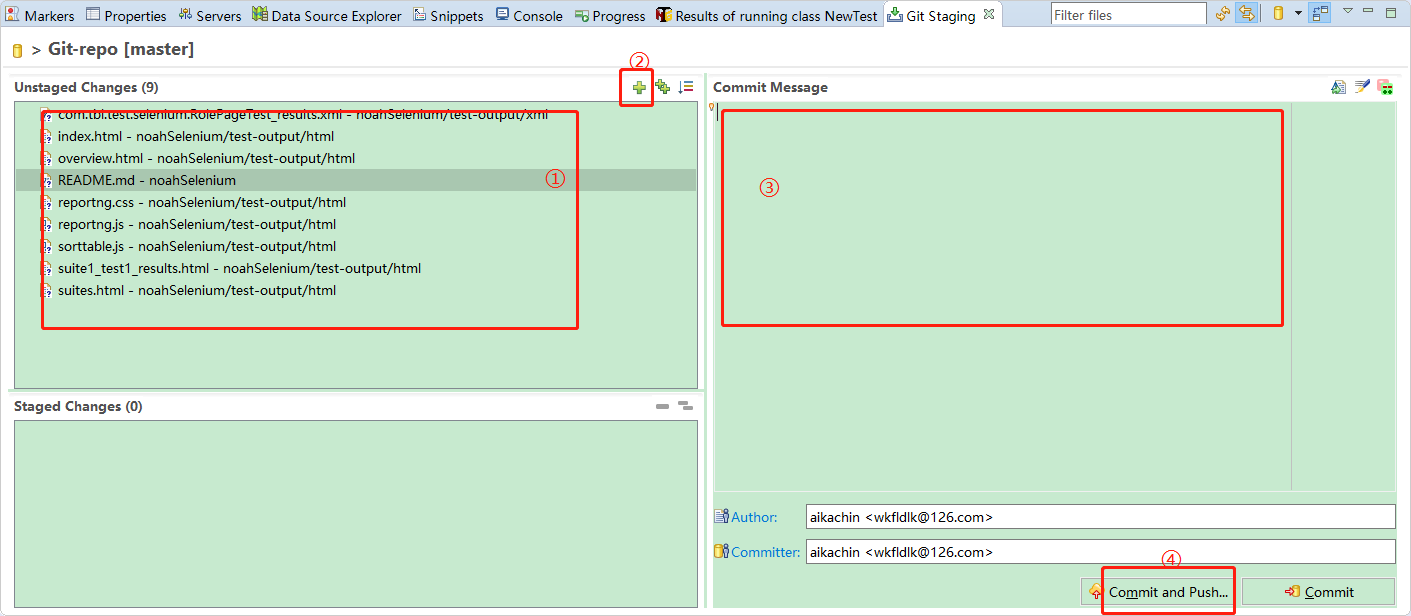


点击Finish，弹出弹框，确认Git的项目地址没问题后，点击close。

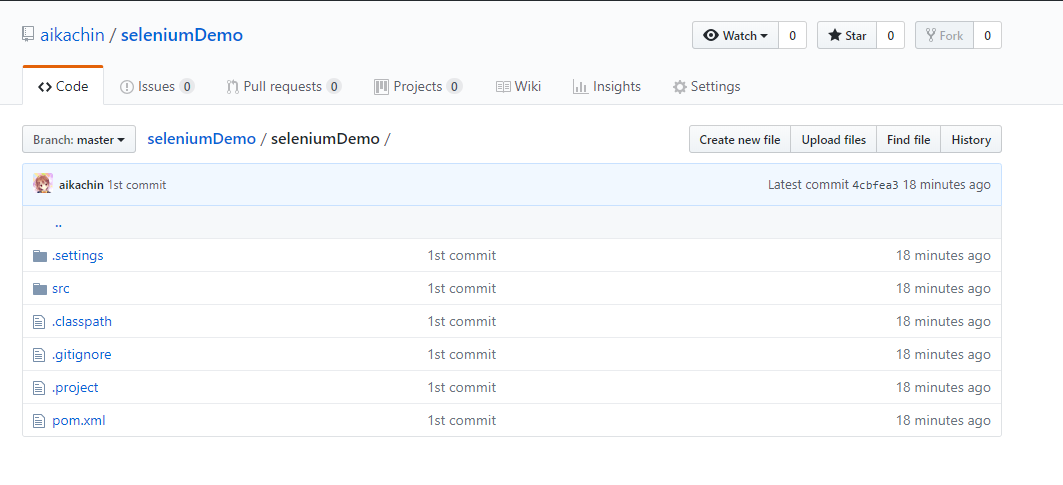


再对着项目右击--Team--Commit

在打开的窗口中按下图顺序依次选择需要上传的文件，点击添加按钮，填写注释，点击Commit and Push即可完成上传到Github。

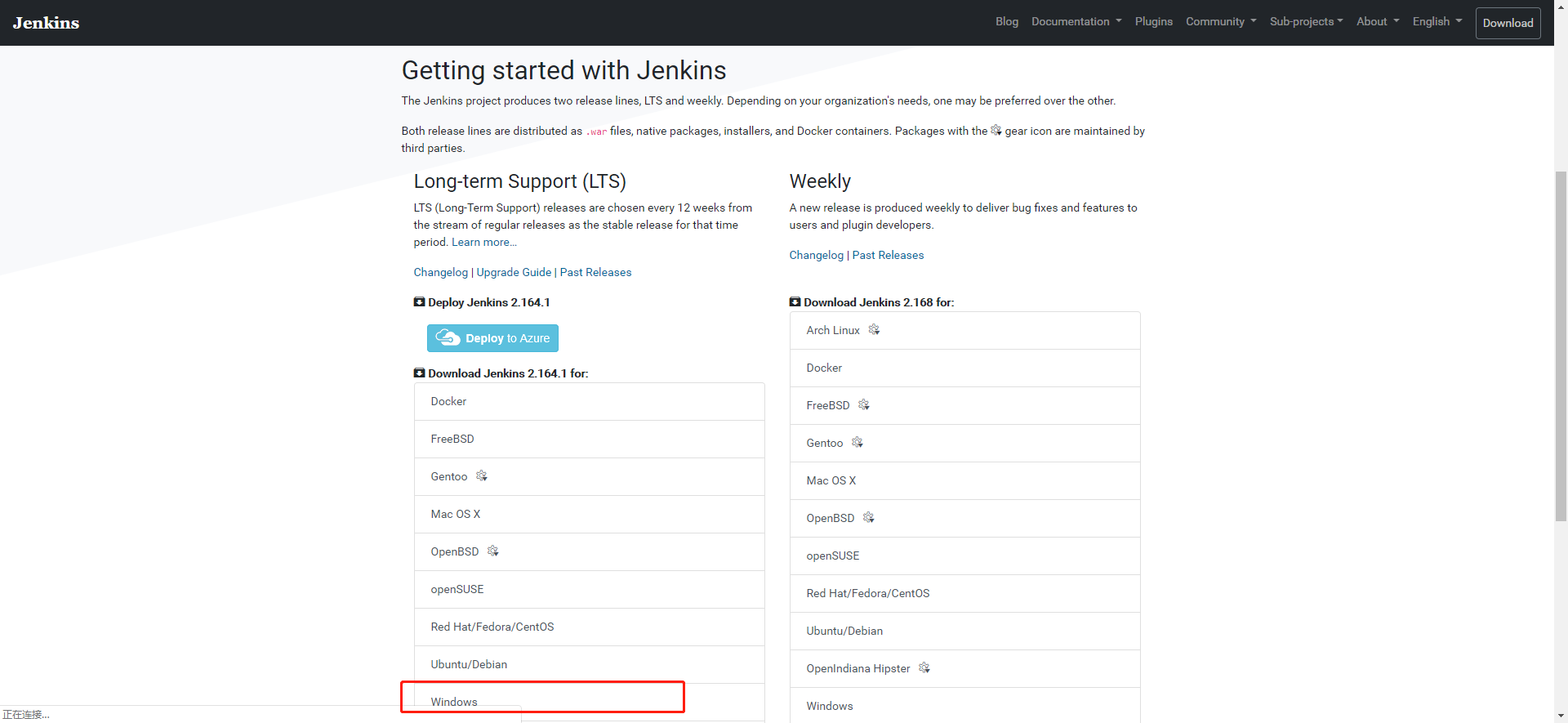


在浏览器刷新GitHub的项目页面，可以发现获取到刚刚上传的内容。



**5.Jenkins安装**

官网下载Jenkins安装包，本例中是使用的安装程序直接安装使用的。



下载后，默认安装完毕后，打开localhost:8080，（如果已经启动其他程序占用端口，也可以尝试修改端口）我没有修改，就用了默认的8080。登录后，提示输入密码，根据页面提示，找到安装目录下的一个初始化密码文件，复制后输入，登录即可。后续，根据提示可以选择安装推荐插件，会花费一些时间。建议安装。然后提示是否添加用户，我默认是尝试添加用户，但是失败了，后来使用admin登录页面空白，。。然后进行了一系列操作，具体不太记得了。

大概包括修改配置文件config.xml，然后登录添加用户组，发现admin也登陆失败，后面又还原了配置文件，直接在浏览器输入localhost:8080/configure后缀的方式可以登录页面，然后就进行一些配置。

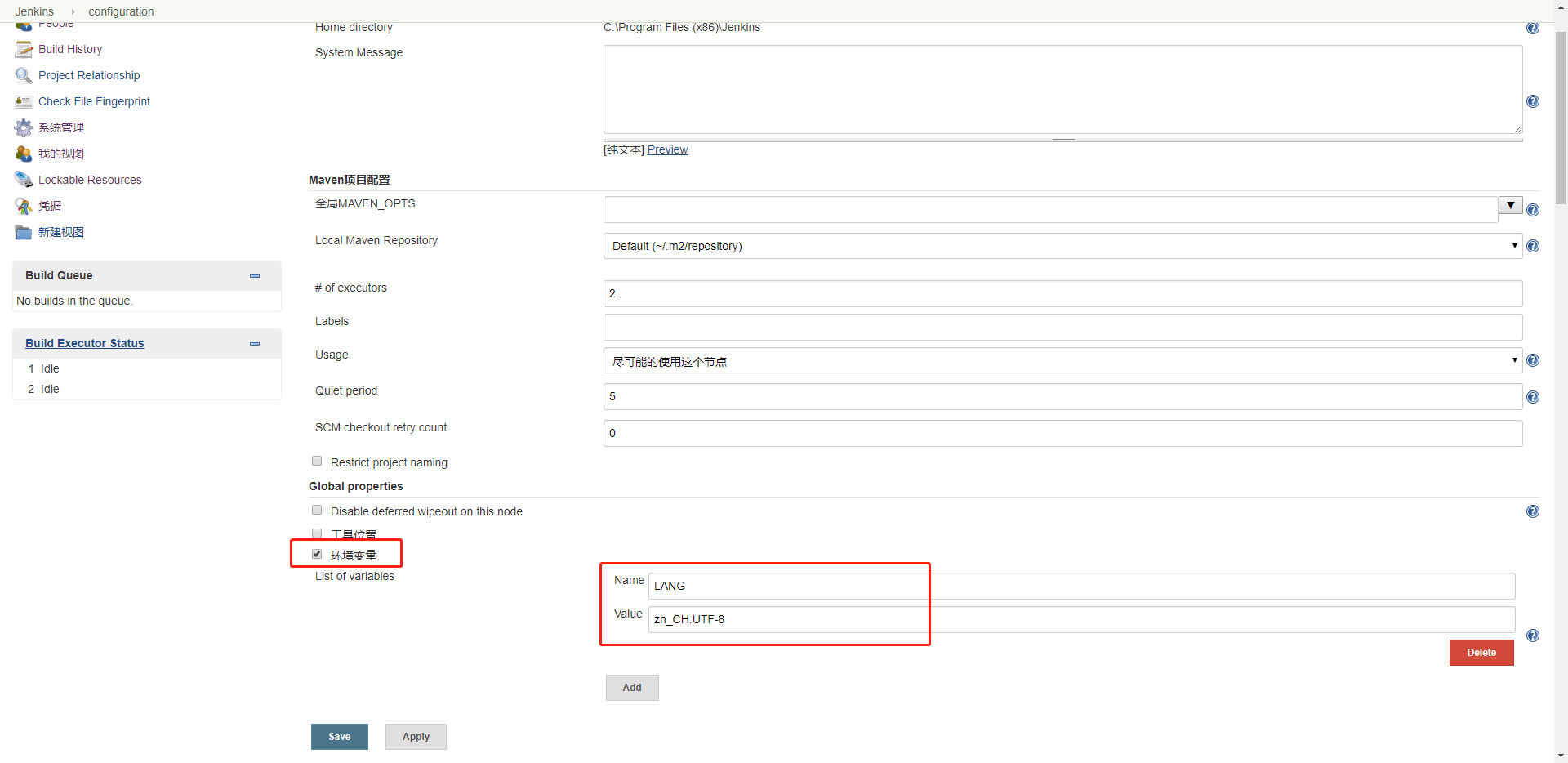
包括：

1）解决控制台终端输出的中文乱码问题

（1）系统设置——配置编码——勾选环境变量，添加如下图所示：

Name：LANG

Value：zh\_CN.UTF-8



(2)在Jenkins安装目录下找到jenkins.xml

打开，添加如下红色部分

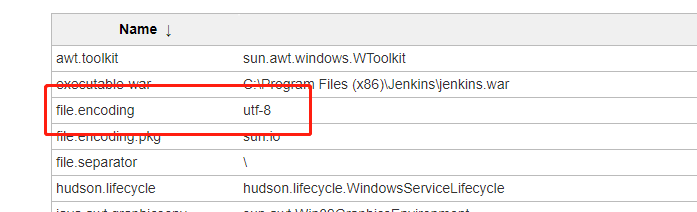
<arguments>-Xrs -Xmx256m **-Dfile.encoding=utf-8** -Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -jar "%BASE%\jenkins.war" --httpPort=8080 --webroot="%BASE%\war"</arguments>

（如果没有这个文件jenkins.xml，需要安装Jenkins为Windows服务：

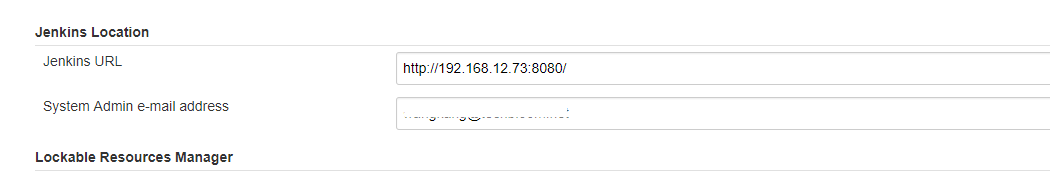
需要在 系统管理-》Install as a Windows Service 将jenkins安装为一个服务。参考：https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+as+a+Windows+service。重启完成后，在用户主目录下即可看到jenkins.xml文件了，本人没有试过，因为默认安装为服务就有这个文件）

上述2步配置编码后需要重启Jenkins。

重启后，进入Jenkins，点击系统管理--系统信息，可以看到编码如下：



另外，配置下图URL后，也可以通过以下URL访问

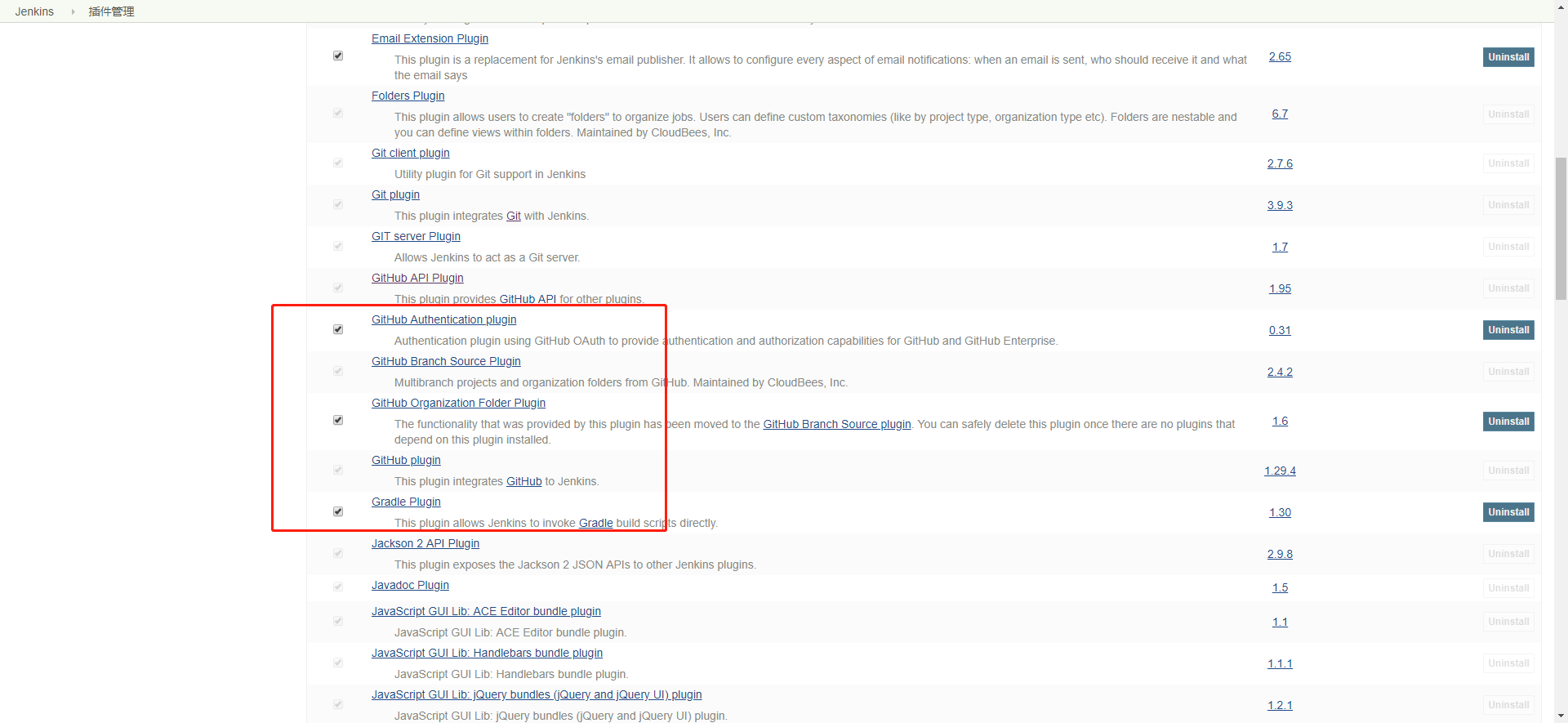


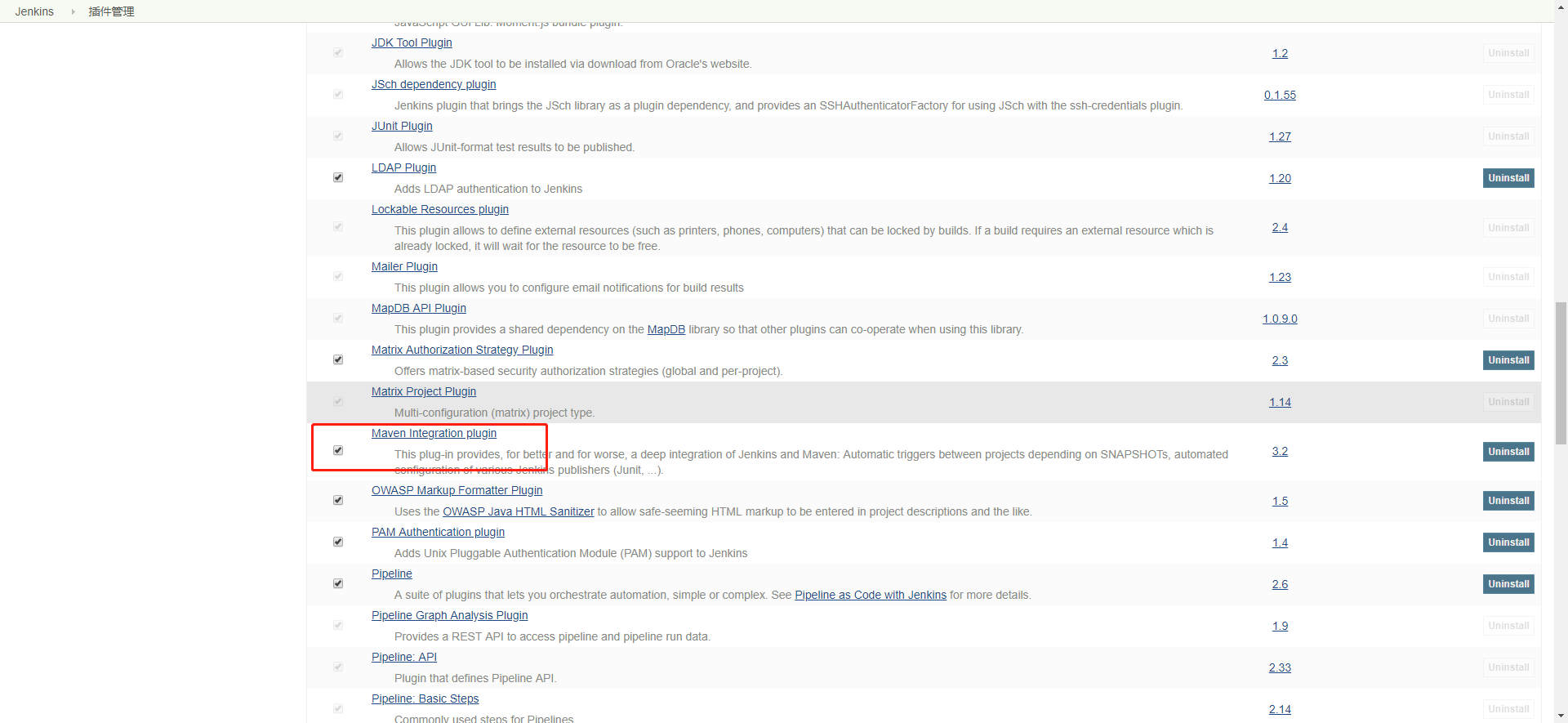
2）插件管理——安装插件

参考：<https://blog.csdn.net/qq_36014509/article/details/80855603>

[https://blog.csdn.net/qq\_39720249/article/details/81264634#%C2%A0%E4%B8%83%E3%80%81%E5%AE%89%E8%A3%85%E5%BF%85%E8%A6%81%E7%9A%84%E6%8F%92%E4%BB%B6](https://blog.csdn.net/qq_39720249/article/details/81264634" \l "%C2%A0%E4%B8%83%E3%80%81%E5%AE%89%E8%A3%85%E5%BF%85%E8%A6%81%E7%9A%84%E6%8F%92%E4%BB%B6)

具体过程之前看过一篇更详细的的介绍文章，但是现在找不到了。。。



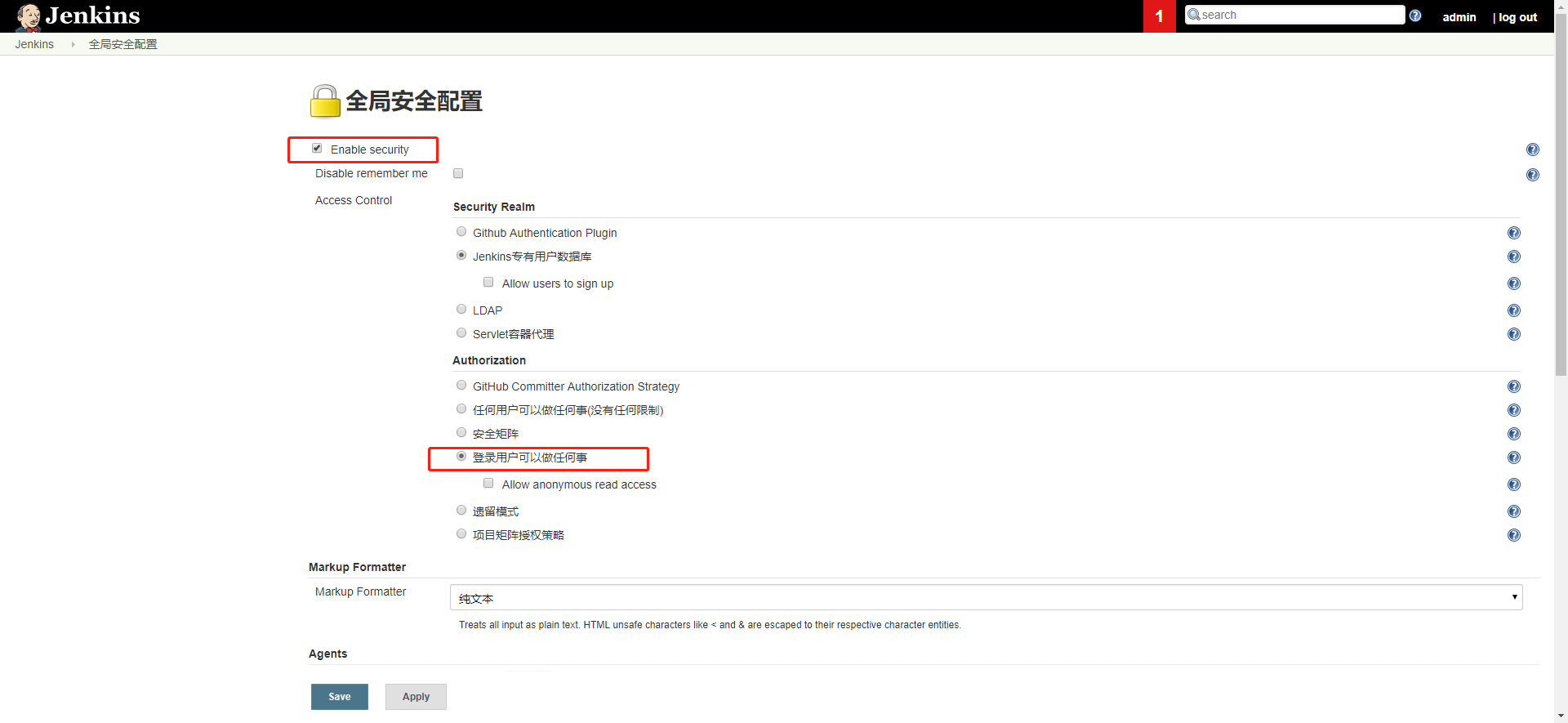


大概安装了这么几个插件：

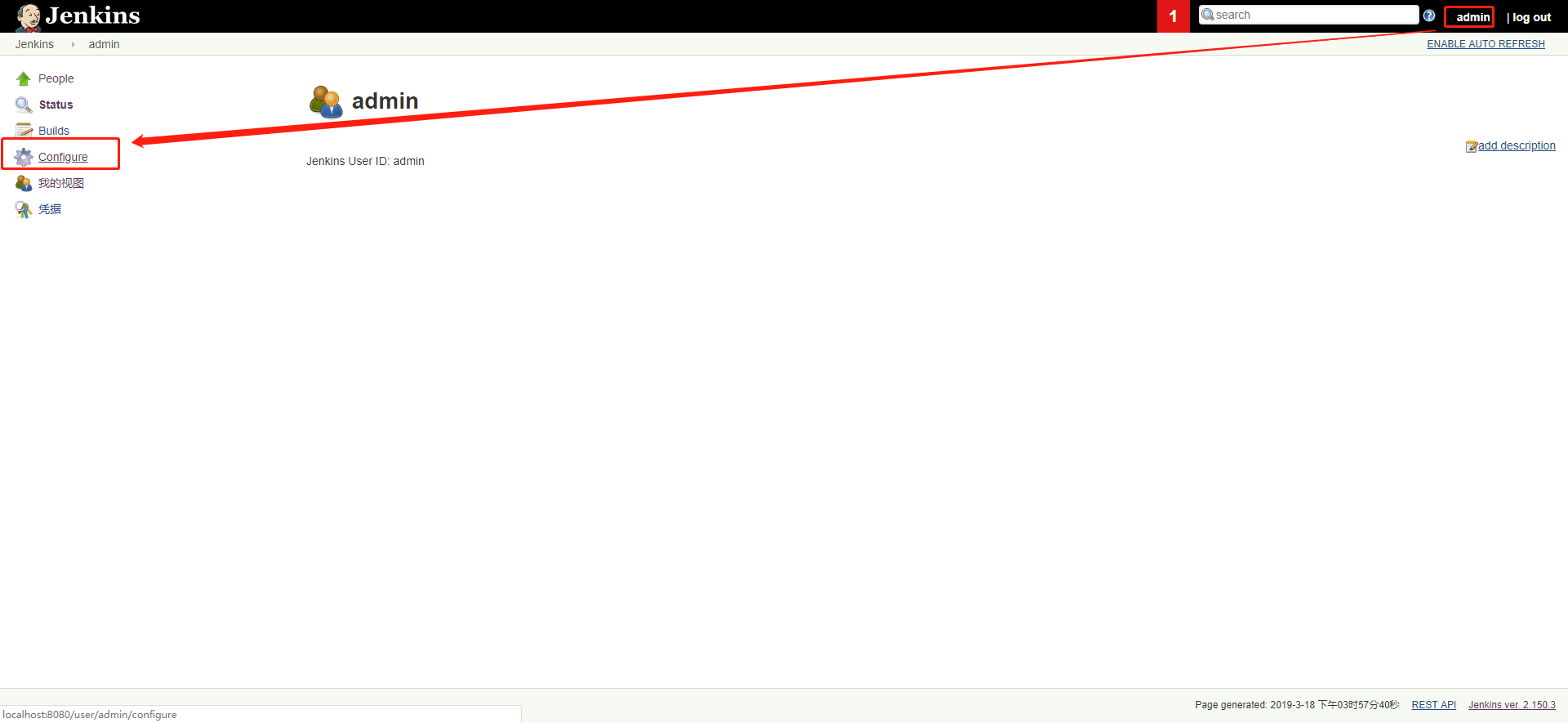


上图参考：[https://blog.csdn.net/qq\_39720249/article/details/81264634#%C2%A0%E4%B8%83%E3%80%81%E5%AE%89%E8%A3%85%E5%BF%85%E8%A6%81%E7%9A%84%E6%8F%92%E4%BB%B6](https://blog.csdn.net/qq_39720249/article/details/81264634" \l "%C2%A0%E4%B8%83%E3%80%81%E5%AE%89%E8%A3%85%E5%BF%85%E8%A6%81%E7%9A%84%E6%8F%92%E4%BB%B6)

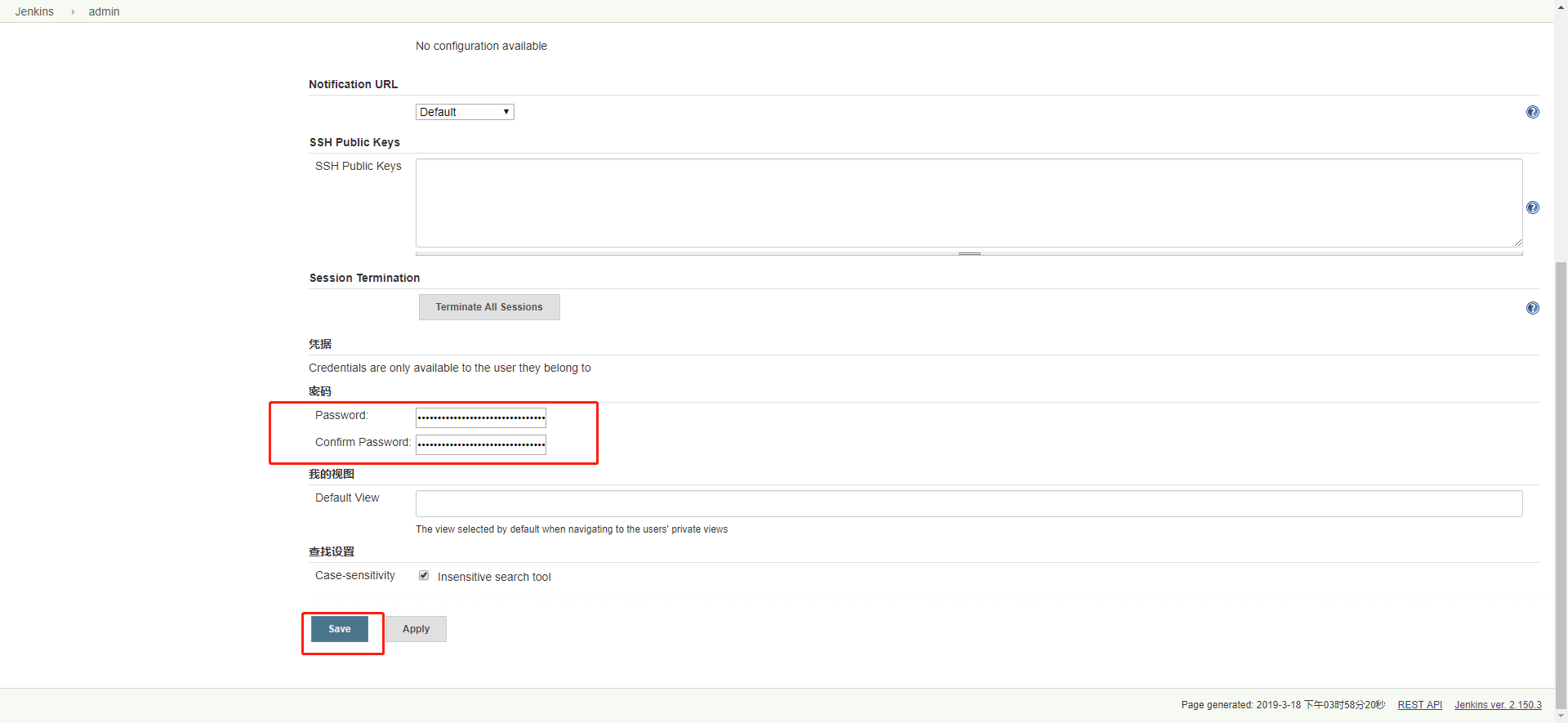
3）配置用户权限（下图偶然发现的，初始安装完后可以配置一下，如果不区分用户组的话）能登录实用就行，此步不一定必要。



4）设置admin密码

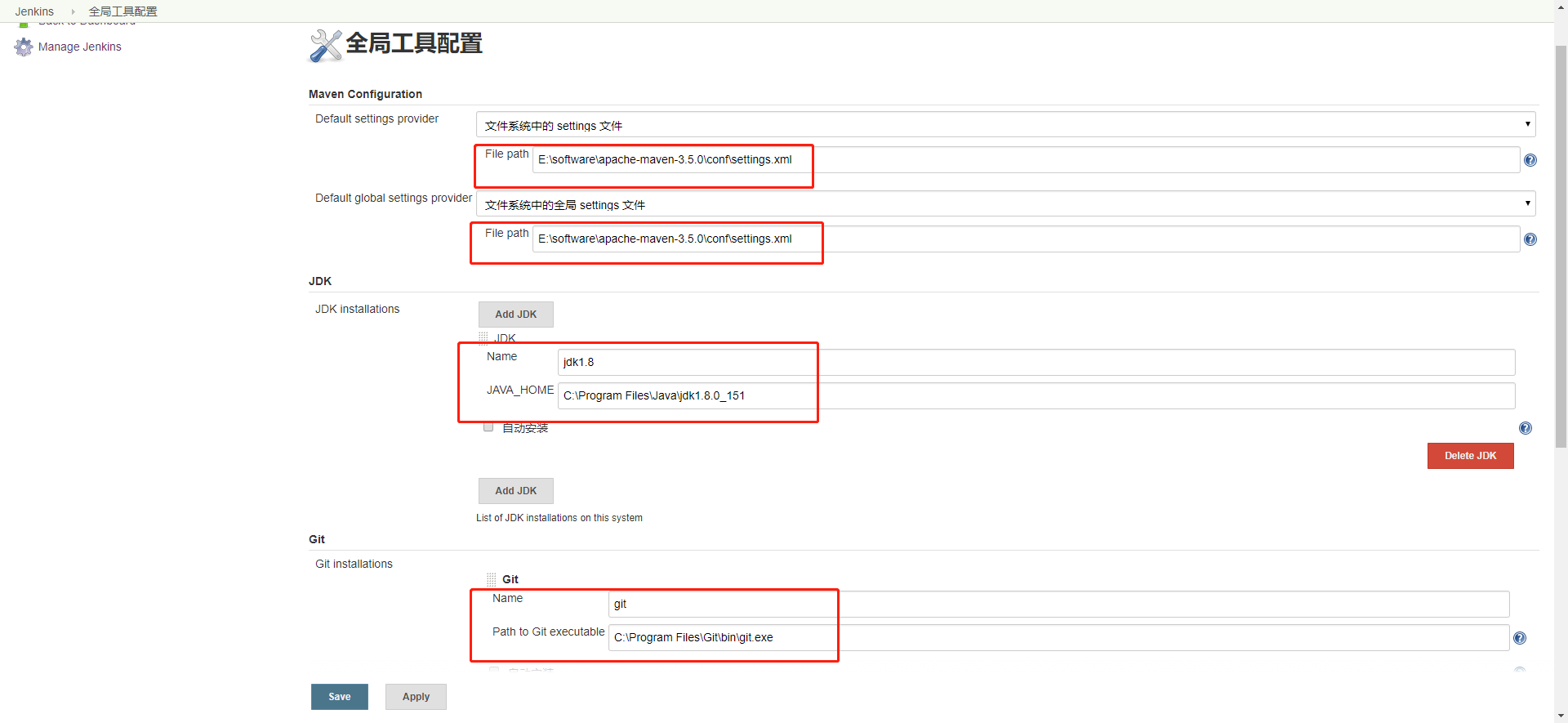


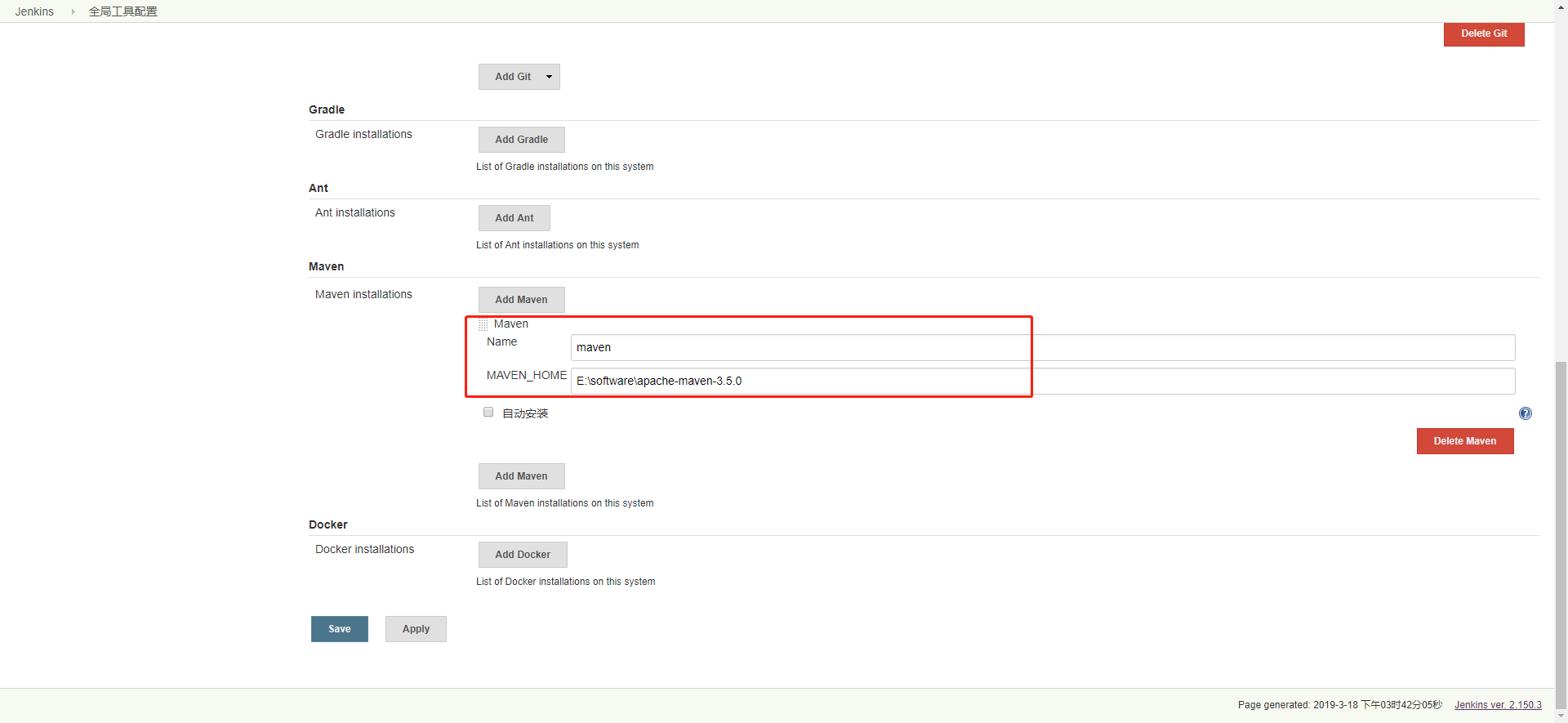
这样配置后登录就方便了。



5）全局工具配置

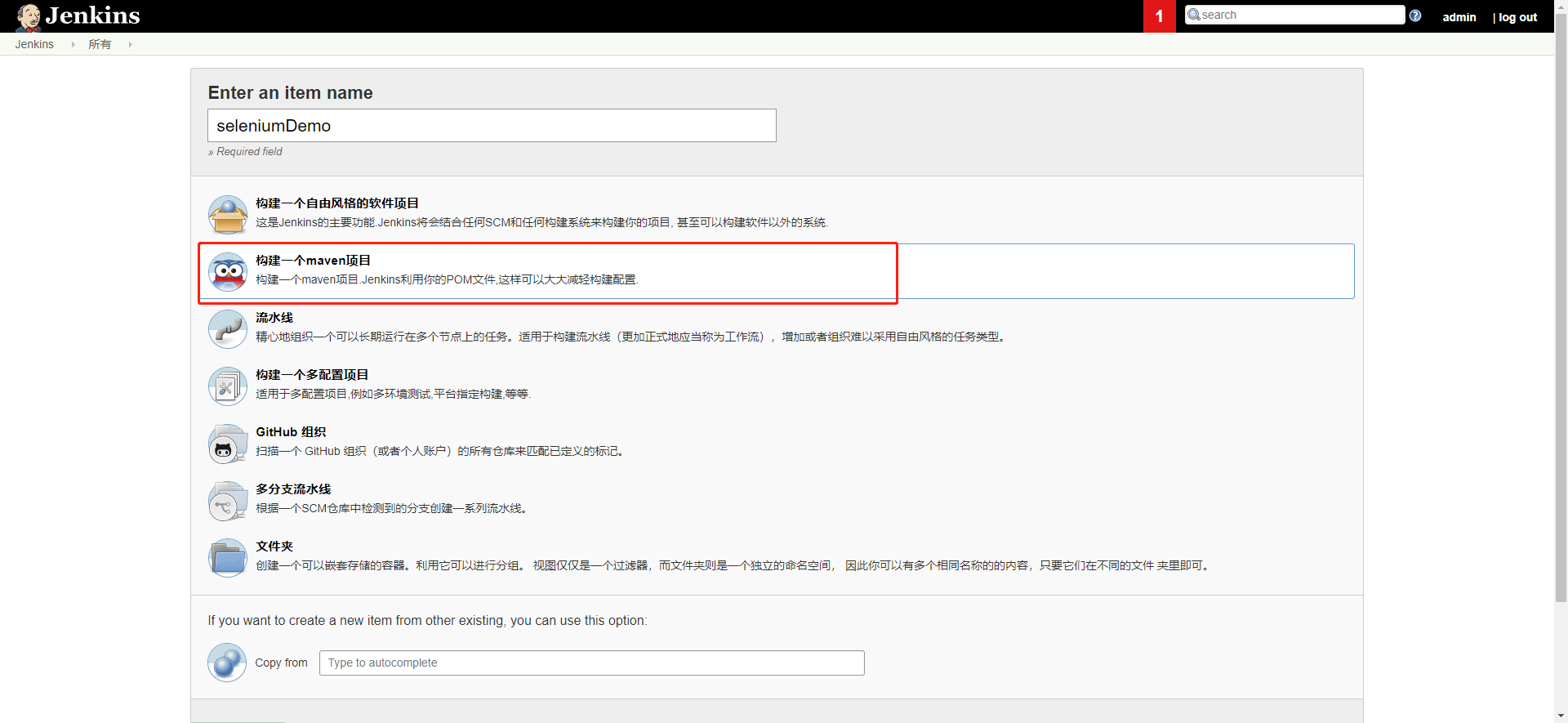
下图配置成自己的实际地址



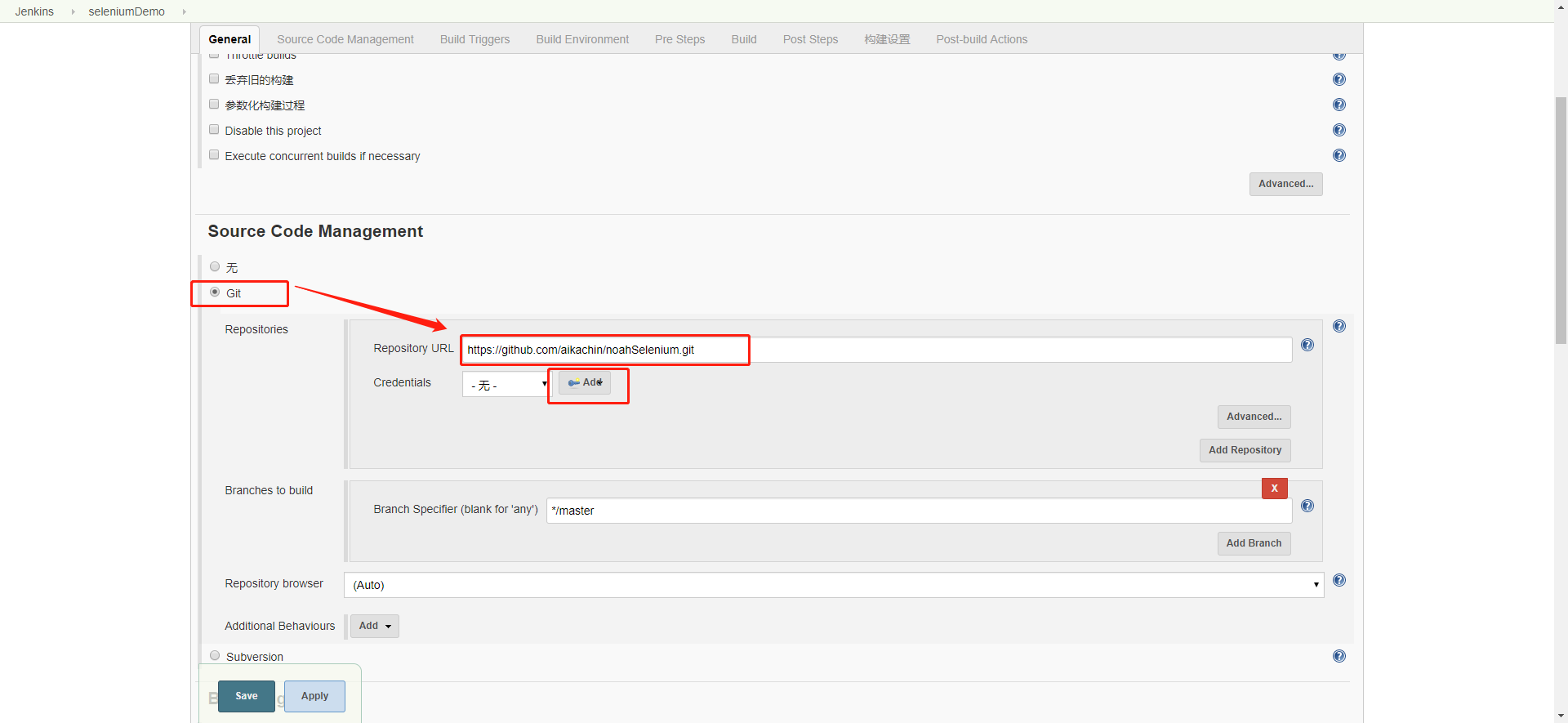


6）接下来进行创建任务

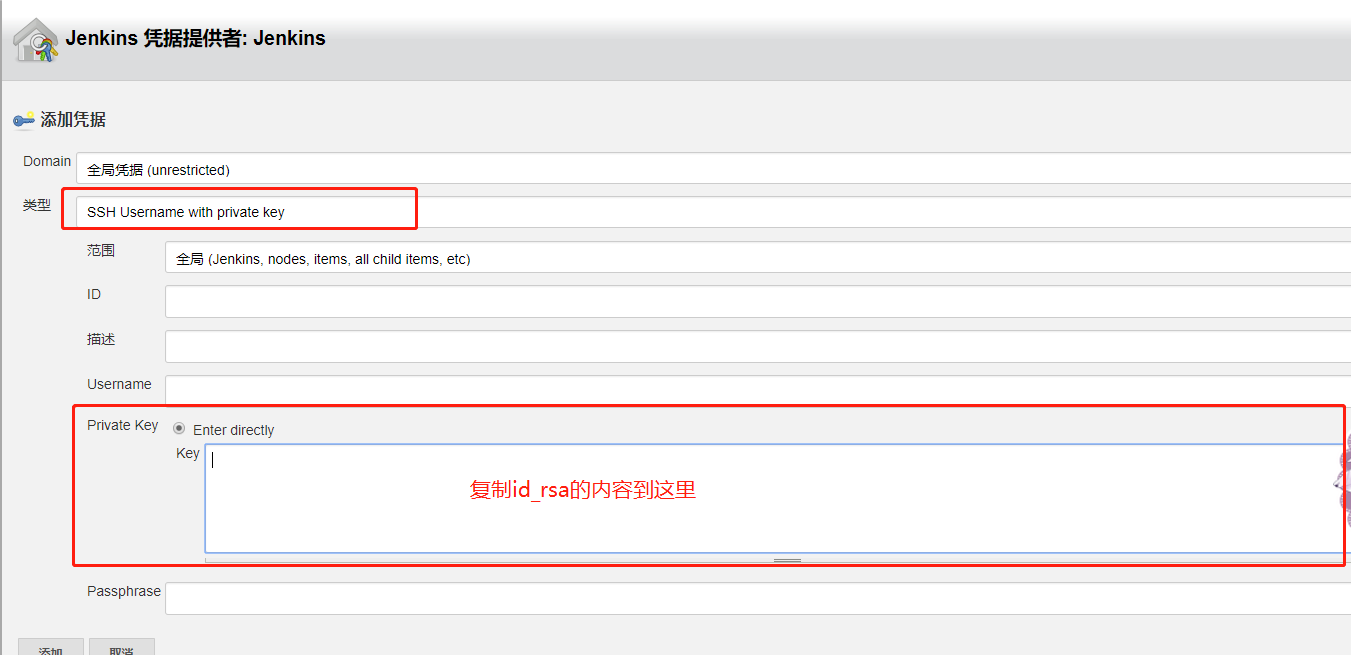
（1）点击创建maven项目



选择Git，输入Git项目地址，Credentials点击Add



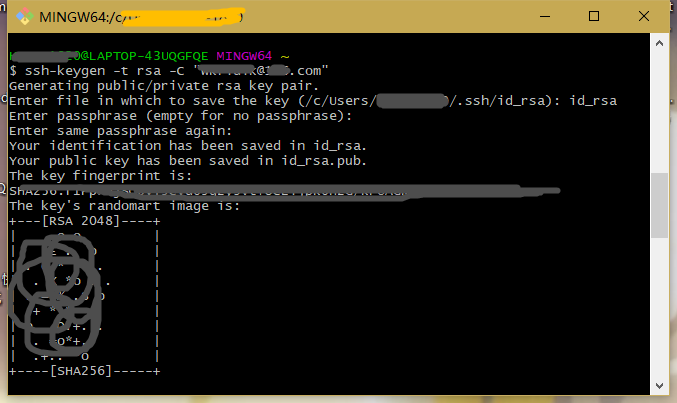
复制下面步骤（2）生成公钥得到的内容到下图中。



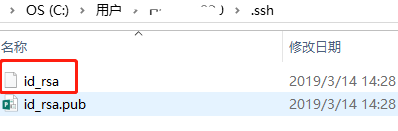
（2）生成公钥，

打开Git Bash，生成公钥，输入

【ssh-keygen -t rsa -C "your\_email@xx.com"】红色部分替换为邮箱地址，然后一直回车，到公钥生成完成。

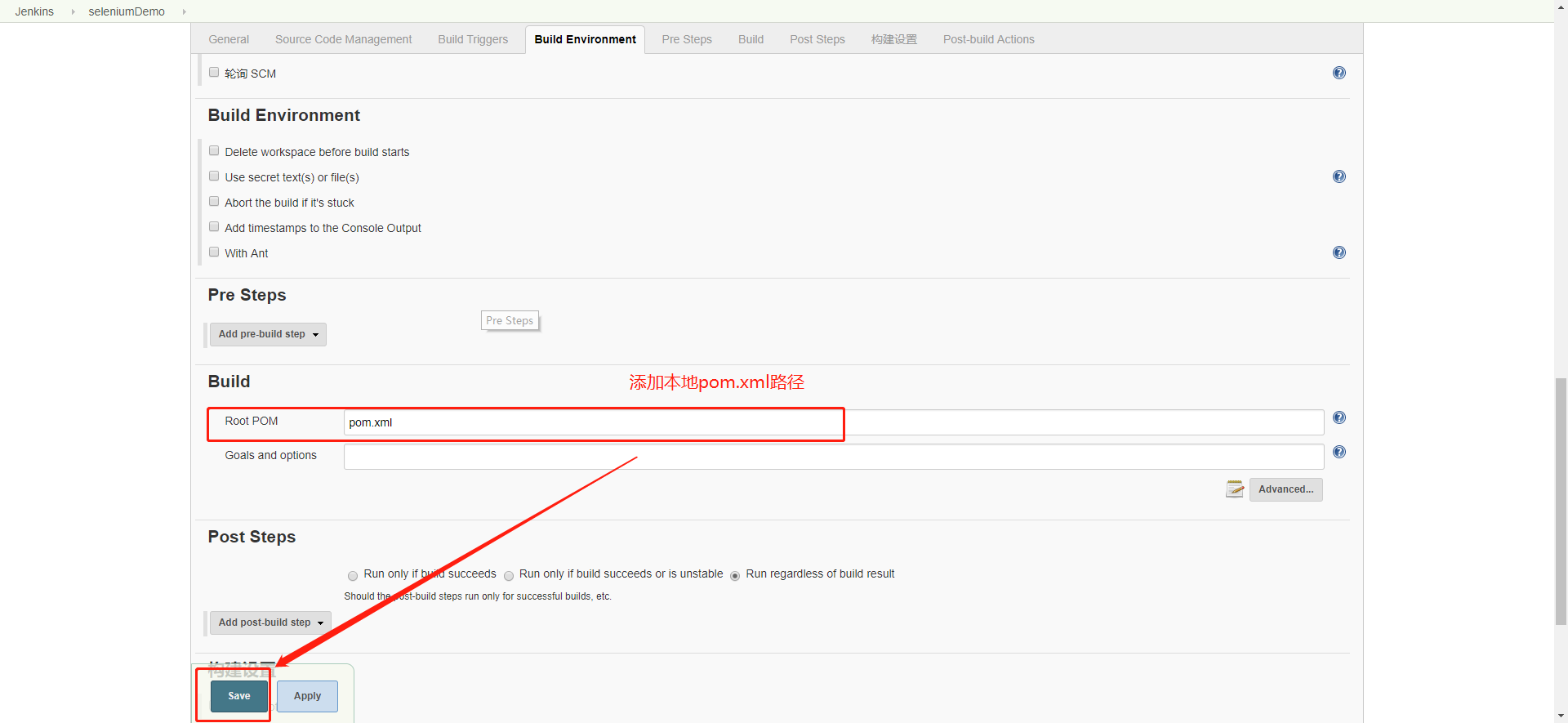


然后在用户目录下的【.ssh】目录可以找到【id\_rsa】文件



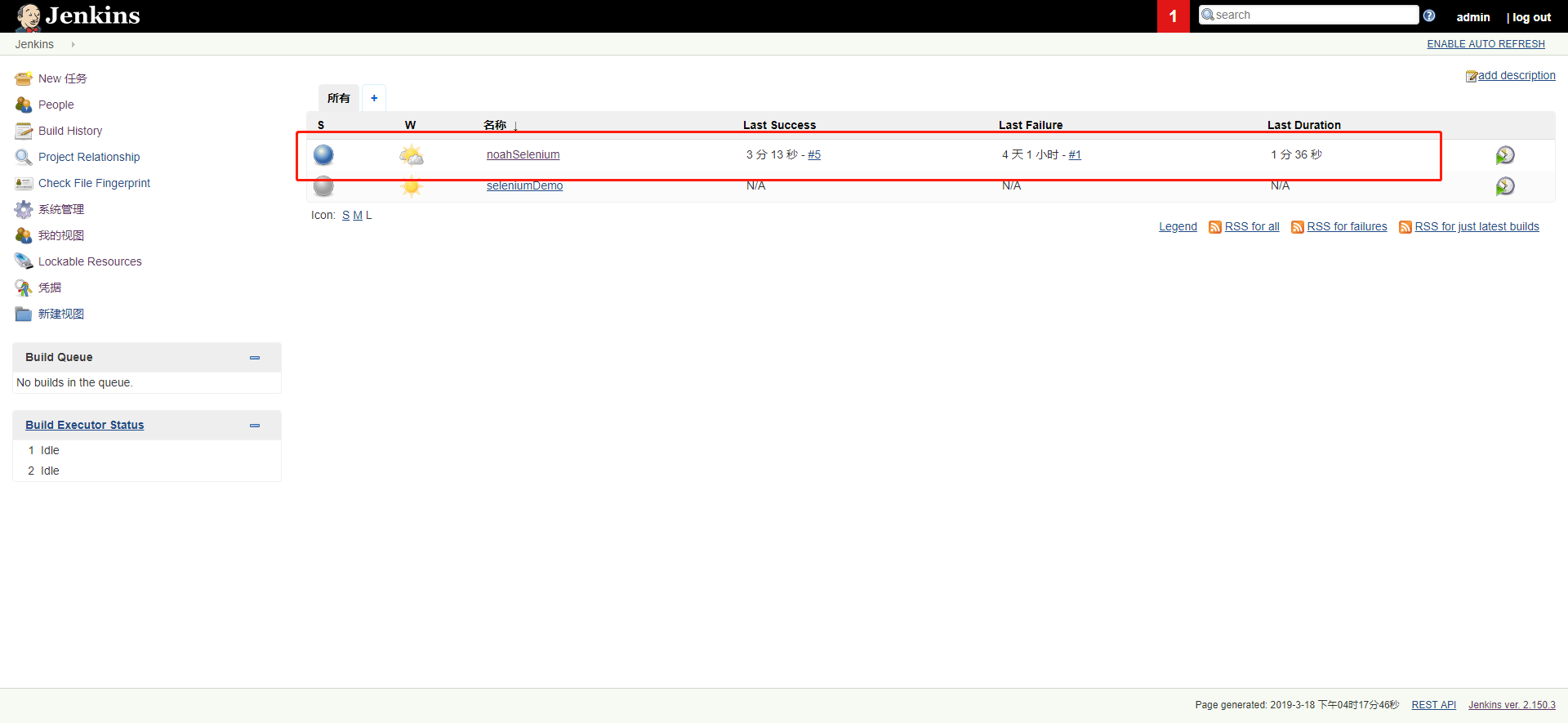
用文本编辑器打开后，复制里面的内容到上面步骤（1）的Private Key框中，点击添加。

然后填写pom.xml路径后点击保存。

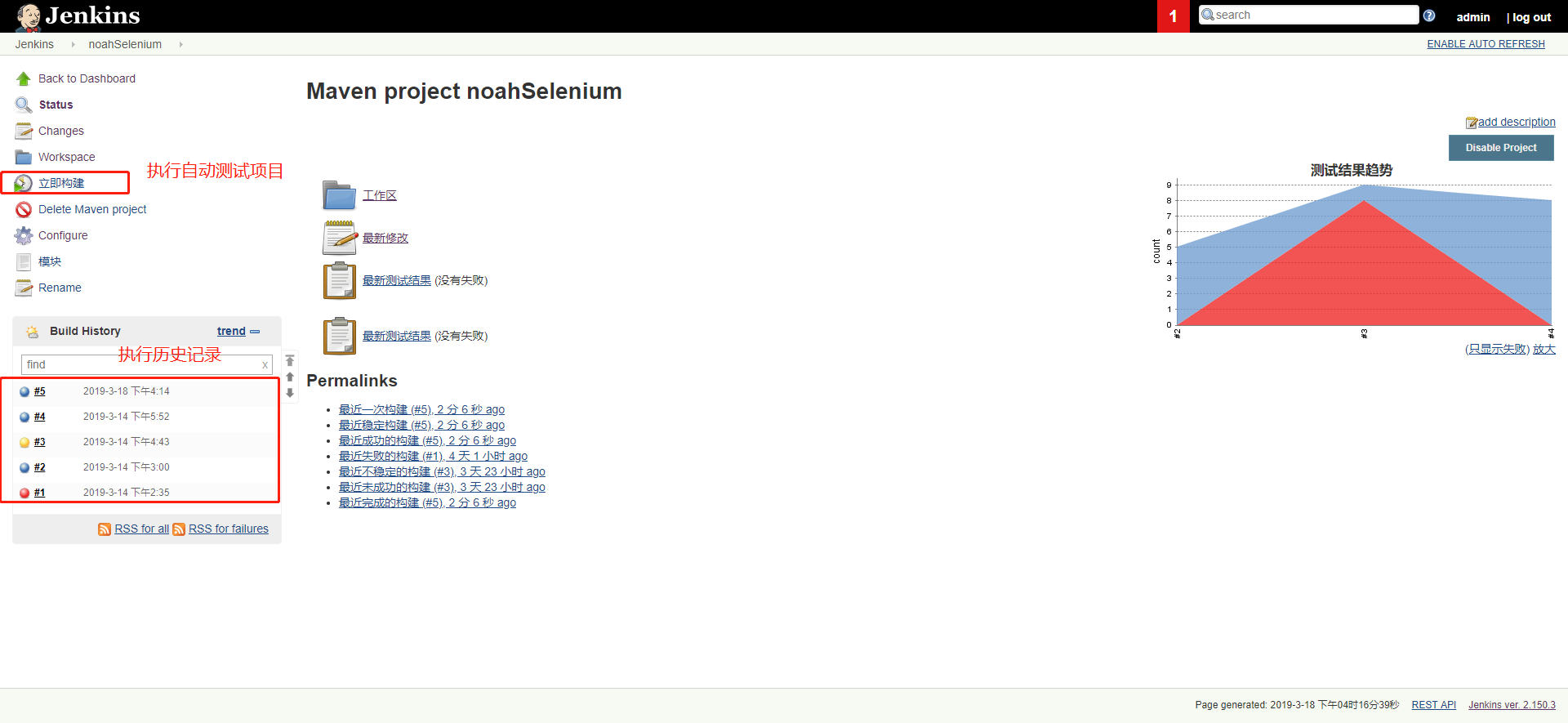


3）构建项目

回到Jenkins首页，可以看到新加的项目，

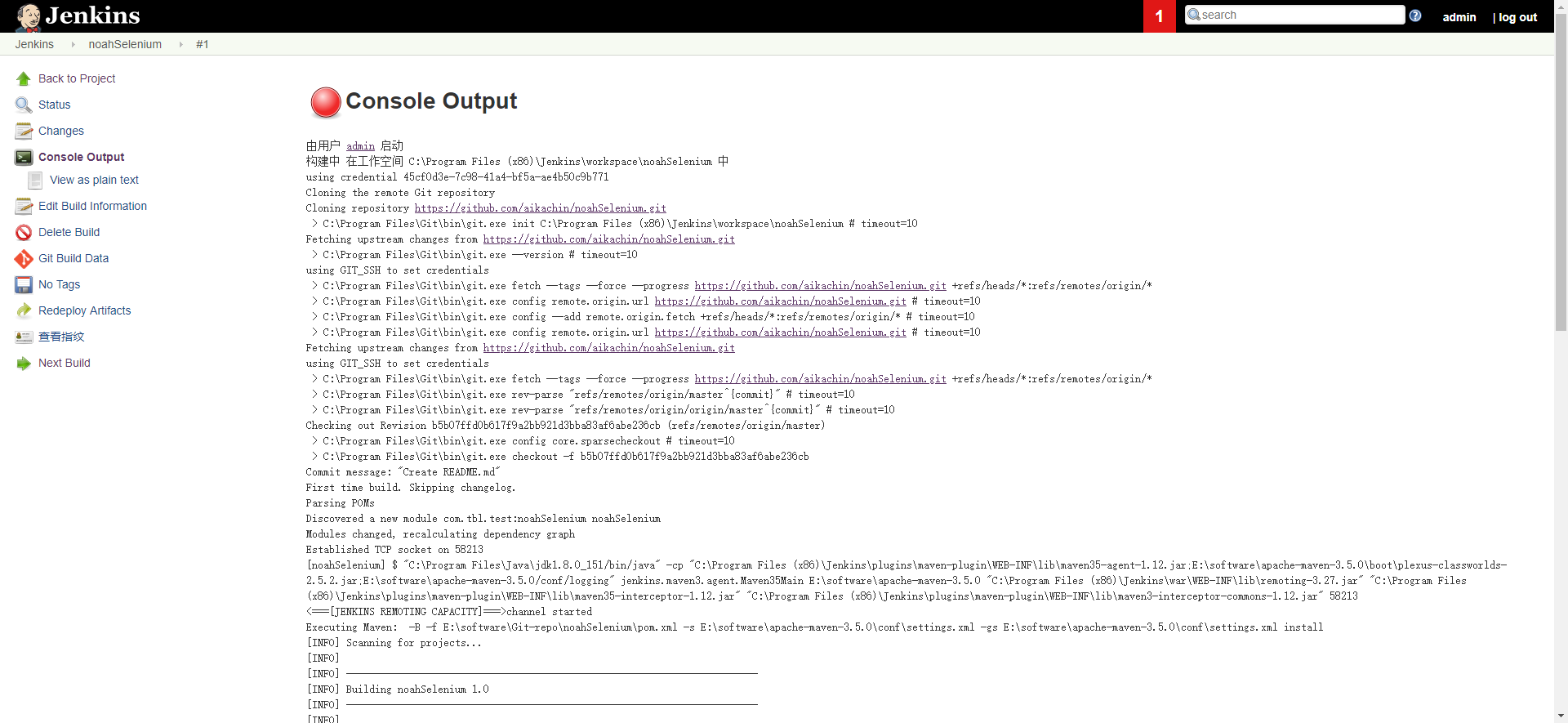


点击项目名称，进入项目详情页面，点击立即构建，就会自动执行项目，如下图。



想要查看及时记录可以点击上图中下面一个红框中的数字左边的圆点。红色表示失败，蓝色表示成功。

点击圆点可以查看控制台输入内容。



构建失败时，会有【Error】标签，通过查看【Error】后面的内容来排查失败原因。

到此，配置暂时告一段落。文首说的一些功能待后面研究回了再继续补充。

\*本文是完成后的一次回顾记录，中途走了不少弯路，可能上述记录会有遗漏。后续发现了再补充。

因为刚开始钻牛角尖了，看到一篇文章说打jar包，然后使用java -jar xxx.jar的方式来运行测试，然后IDEA下maven项目的testNG依赖默认只支持test测试代码下的依赖调用，main源代码里面的无法获取到testNG的jar包的依赖，一致报错，build不了，而打成jar又默认只能运行main中的代码；然后又在使用IDEA还是eclipse中纠结了好久，想着2种都搞一下，结果浪费了好久时间。

事实证明，方式不止一种，从eclipse里面跑通的testNG和maven之后，后续工作就安装Jenkins，和更新到Github 两个环节，然后Jenkins添加项目，配置Git，构建就完了。并没有想象中的麻烦。

是我一开始想太多了，思维定势被牵着鼻子走了。-\_-|| 以后要注意这种情况。

2019-3-18完稿