# Git学习

## Windows 10 配置Github SSH Key

1）安装好以后打开：Git Bash，执行：

$ ssh-keygen -t rsa -C [drbjhu@163.com](mailto:drbjhu@163.com)

然后点击回车，设置密码，也可设置为空。一路确定！

2） 然后进入当前的用户的目录下，打开 .ssh 文件夹里面会有2个文件： id\_ras 和 id\_rsa.pub

C:\Users\Administrator\.ssh

3） 登录GitHub系统；点击右上角账号头像的“▼”→Settings→SSH and GPG keys →New SSH key。

4）复制id\_rsa.pub的公钥内容。添加到Key 中，Titile可以随便填写（e.g., HBJ-PC）。最后 Add SSH Key。

5） 测试ssh keys是否设置成功。

$ ssh -T [git@github.com](mailto:git@github.com)

The authenticity of host ‘github.com (192.30.253.112)’ can’t be established.

RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

Are you sure you want to continue connecting (yes/no)? yes #确认你是否继续联系，输入yes

Warning: Permanently added ‘github.com,192.30.253.112’ (RSA) to the list of know n hosts.

Hi hugebawu! You’ ve successfully authenticated, but GitHub does not provide shell access. #出现这句话，说明设置成功。

## 常用命令

# git status -s(-s 表示获取简短的结构输出) #查看当前项目的当前状态"M":已修改；"??":未添加，"A":已添加到缓存，"AM":这个文件在我们将它添加到缓存之后又有改动。

$ git branch -a #查看项目的分支们(包括本地和远程)

#git branch <name> 创建分支

#git checkout/(switch -c) 分支名 #切换分支

#git checkout -b 新分支名 #添加新分支

#git branch -d 分支名 #删除本地分支

# git push origin --delete <BranchName> #删除远程分支

# git diff #执行 git diff 来查看执行 git status 的结果的详细信息。git diff 命令显示已写入缓存与已修改但尚未写入缓存的改动的区别。git diff 有两个主要的应用场景；尚未缓存的改动：git diff；查看已缓存的改动： git diff –cached；查看已缓存的与未缓存的所有改动：git diff HEAD；显示摘要而非整个 diff：git diff –stat

## 获取源码

#ssh-keygen #生成公私钥命令

#git clone -b v1.1.0 [git@github.com:hyperledger/fabric-sdk-java.git](mailto:git@github.com:hyperledger/fabric-sdk-java.git)

#git checkout -b origin/release-1.1不能用git checkout -b v1.1.0，否则会maven Update Project出错。

## 本地代码提交到远程仓库

将本地代码上传到远程仓库的时候，打开命令行窗口，进入到本地代码的文件夹。

#rm -rf .git\* #清理旧的git相关文件

$ git init #Git 使用 git init 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 git init 是使用 Git 的第一个命令。

在执行完成 git init 命令后，Git 仓库会生成一个 .git 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变（不像 SVN 会在每个子目录生成 .svn 目录，Git 只在仓库的根目录生成 .git 目录）。

如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交：

$ echo "# fabric" >> README.md #新建README.md文件并输入

$ git add README.md

但这一步是添加README.md文件，初始化git，这样在本地文件夹下会生成.git文件夹。具体这个文件是做什么的，可以自行百度，不过一句话概括的话，是将本地仓库与远程仓库关联起来。

$ git add . #将当前项目的所有文件添加到缓存区 #

$git commit -m "first commit" #将缓存区内容添加到仓库中 （提示出错则需要配置用户名和邮箱地址 $ git config --global user.name hugebawu

$ git config --global user.email drbjhu@163.com）

到commit这一步，只是完成了本地提交。这里为什么有一个add一个commit的过程，有必要解释一下。在git中有一个暂存区的概念，add就是将代码从工作区提交到暂存区；commit的过程就是将代码从暂存区提交到版本库。git add . 是提交全部文件到暂存区，git commit -m后面相当于给本次提交加一个注释。

这里解释一下git为什么有暂存区的概念，这是和SVN所不同的。git是分为工作区和版本库的，工作区就是我们本地看到的工作目录，版本库就是你工作目录中的.git文件。

工作区：就是你在电脑里能看到的目录。

暂存区：英文叫stage, 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。

版本库：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

--------------------------------

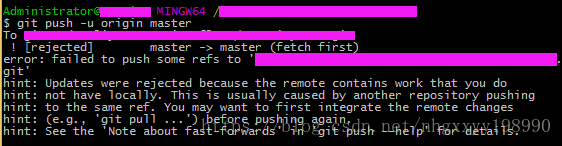
版本库中存放了很多东西，比如stage,就是我们上面提到的暂存区，还有自动创建的master分支，还有指向master的head指针…….我们不能手动修改版本库内的任何数据，否则可能会造成不可预期的错误。所以我们上面，将工作区的代码提交到版本库是分为两个步骤的，第一步是通过add命令将文件添加到暂存区stage中；第二步将暂存区中的数据提交到版本库中，用commit命令，其实就是将stage中的内容提交到当前分支。我们在创建git版本库时，会为我们自动创建一个master主分支(#git branch -a来查看)，当然我们还可以手动创建其他分支，当前分支是什么，我们commit的时候就将stage的内容提交到该分支上。

在github个人网页手动新建fabric-sdk-java仓库

#git remote add origin [git@github.com:hugebawu/fabric-sdk-java.git](mailto:git@github.com:hugebawu/fabric-sdk-java.git)(第一执行才需要)

$git push -u origin release-1.1 #push到远程仓库#最后一个填本地想要提交的分支名(可以通过git checkout -b <new branch name>在本地仓库创建新分支)。

最后将版本库中的代码提交到远程分支上去。到这里，将本地代码提交到远程仓库的步骤就全部完成了。我在提交的时候也遇到一些问题，拿出来我们一起分享下



出现错误的原因是git服务器中的README.md文件不在本地代码目录中，可以通过如下命令进行代码合并

git pull --rebase origin master

执行上面的代码就可以使本地和git服务器同步了。其实如果有时间，可以好好研究一下这个.git和README.md文件。.git文件里面存放的内容上面简单介绍了一下，README.md文件要研究起来内容还是蛮多的，这里推荐一个博客感觉介绍的挺好的，感兴趣的源儿们可以看一下原来Github上的README.md文件这么有意思——Markdown语言详解

好了 这篇博客就介绍到这里，顺便推荐一下廖雪峰关于git的文章，介绍的非常好，非常适合初学研究，整体看下可以形成一个宏观的知识体系。好的，先这样，后续再更新。

# maven的安装和环境配置

## 什么是maven?

Maven是一个项目管理工具，它包含了一个项目对象模型 (Project Object Model)，一组标准集合，一个项目生命周期(Project Lifecycle)，一个依赖管理系统(Dependency Management System)，和用来运行定义在生命周期阶段(phase)中插件(plugin)目标(goal)的逻辑。当你使用Maven的时候，你用一个明确定义的项目对象模型来描述你的项目，然后Maven可以应用横切的逻辑，这些逻辑来自一组共享的（或者自定义的）插件。

Maven 有一个生命周期，当你运行 mvn install 的时候被调用。这条命令告诉 Maven 执行一系列的有序的步骤，直到到达你指定的生命周期。遍历生命周期旅途中的一个影响就是，Maven 运行了许多默认的插件目标，这些目标完成了像编译和创建一个 JAR 文件这样的工作。

一个jar包,会有两种下载的方式。一种是直接下载压缩包，一种是提供maven下载路径。Maven最方便的就是能帮我们下载jar包。告别传统手动导包的方式。

## maven仓库

maven中有中央仓库，本地仓库，私服三个概念

1.中央仓库是maven给你提供的所有jar包的下载地址.网址：http://mvnrepository.com/

2.本地仓库是你在中央仓库里下载好的jia包所保存的文件夹。

3.私服相当于一个大型的本地仓库，一般在规模庞大的公司里才会有自己的私服。为什么会有私服呢？ 有些公司在上班过程中某些电脑是不允许连接互联网的，只能使用自己公司的局域网，那么，这个时候你要使用maven的话就不能使用http的地址，这个时候就产生了私服。私服通过maven的索引使用公司的ip地址来使用到maven中的jar包。中国拥有最厉害的私服公司就是马云的阿里巴巴。但是阿里巴巴的私服会对外开放。

## 下载maven  apache-maven-3.3.9-bin

安装maven之前必须要安装好java版本1.6以上

<http://maven.apache.org/download.cgi>

## 解压 环境配置

### 解压你在maven下载到的压缩包，一般把它和jdk放在一起。

e.g., C:\Program Files\Java\apache-maven-3.6.2

### 添加系统环境变量

maven\_home的变量值是你maven的压缩包的解压地址

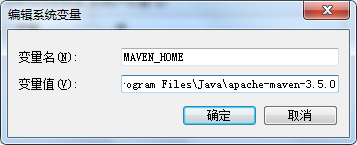
MAVEN\_HOME : C:\Program Files\Java\apache-maven-3.6.2

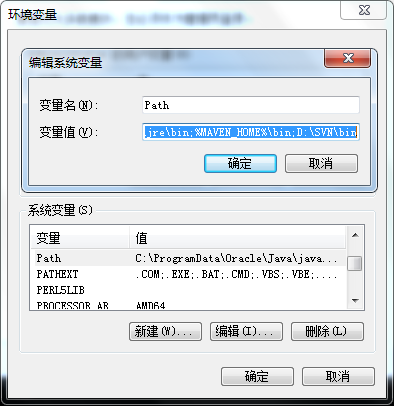
MAVEN :  %MAVEN\_HOME%\bin

MAVEN\_OPTS :  -Xms256m -Xmx512m (这两条貌似不需要)

在Path里添加%MAVEN\_HOME%;

%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;后面两个也必须保证有。





这个也需要CLASSPATH: .;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar

## 环境变量配置好后，查看Maven环境版本使用cmd命令：mvn –version

## Maven的使用

进入项目根目录

### #mvn compile

compile是maven工程的编译命令，作用是将src/main/java下的文件编译为class文件输出到target目录下

### #mvn test

这一步的正确执行需要按照搭建好fabric环境，相关的容器得跑起来。

test是maven工程的测试命令，会执行src/test/java下的单元测试类。

### #mvn clean

clean是maven工程的清理命令，执行 clean会删除target目录的内容。

### #mvn package

package是maven工程的打包命令，对于java工程执行package打成jar包，对于web工程打成war包。

# Eclipse配置

<https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2019-09/R/eclipse-jee-2019-09-R-linux-gtk-x86_64.tar.gz>

## 我的常用快捷键

Ctrl+Shift+T 打开类型

Ctrl+Shift+R 打开资源(查找目标文件)

**搜索**

Ctrl+H 全局搜索

Ctrl+G 工作区中的声明

Ctrl+Shifi+G 工作区中的引用

Ctrl+T 查看某一个类的继承类或者实现类

**文件**

Ctrl+Shift+S 全部保存

Ctrl+Shift+F4 全部关闭

Alt+Enter 属性

Ctrl+N 新建

源代码

Ctrl+Shift+Y 将所选区域字母设置为小写

Ctrl+Shift+X 将所选区域字母设置为大写

Alt+Shift+J 方法添加注释

运行

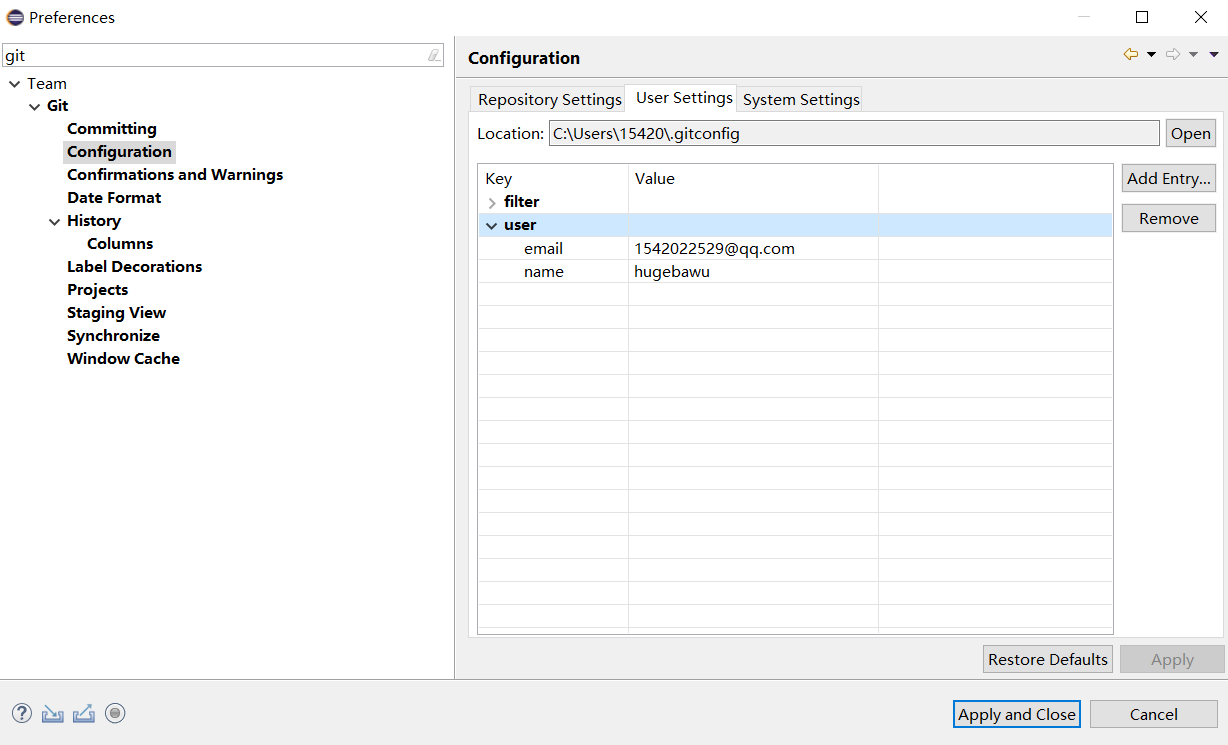
Alt+R Run Junit Test(windows>>Preferences>>Keys中自定义配置)

Ctrl+F11 运行上次启动

## Eclipse配置git

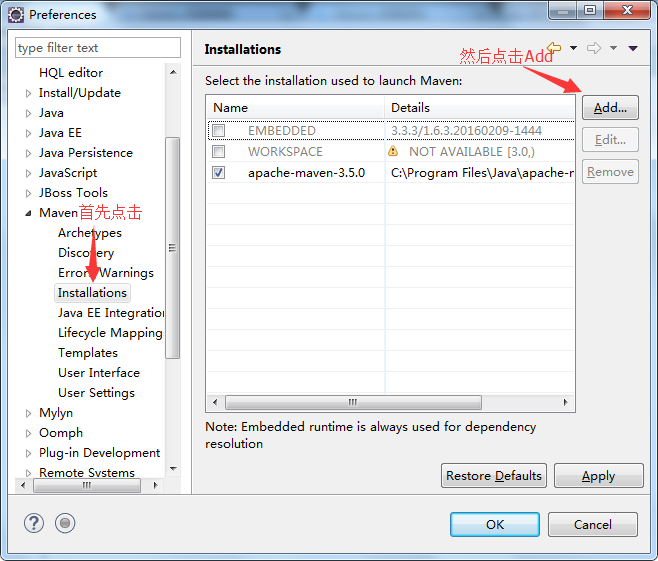
这个“提交者”不是用来与git服务提供商进行权限认证的，而是用来在提交的时候说明“是谁”提交的，可以这个时候指定，也可以第一次提交的时候通过自动向导来指定，我就是通过第一次提交的时候输入的，下面来看一下保存的信息，顺便可以直接这个时候就指定了。

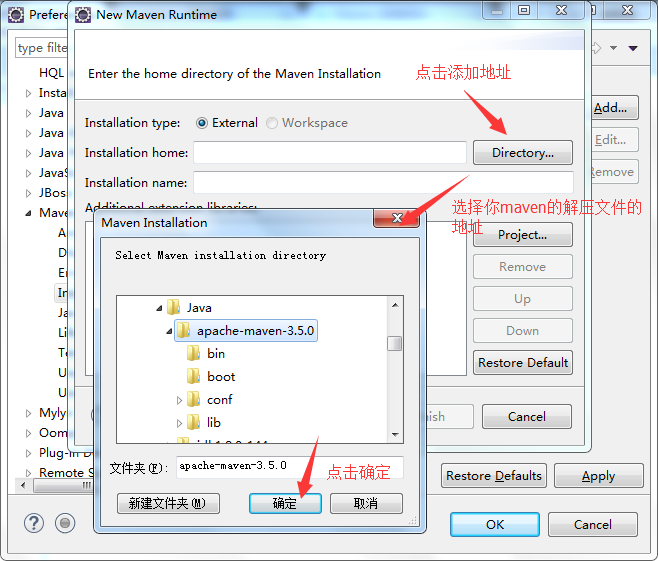
展开“git”，点击里面的“configuration”，右边就是全局的设置项，点击“add entry”添加项目，使得添加后的形式如下即可。



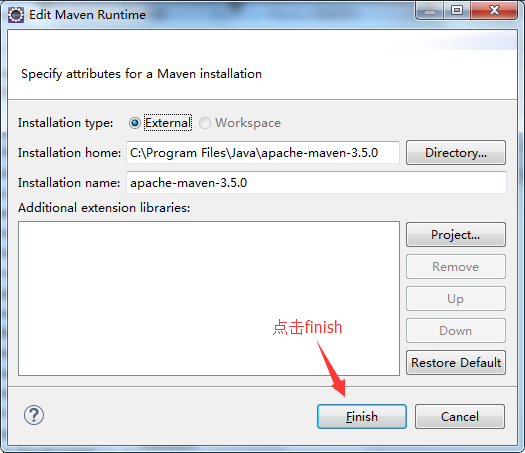
## 在Eclipse上安装maven

### 打开Eclipse点击window>prferences之后会弹出

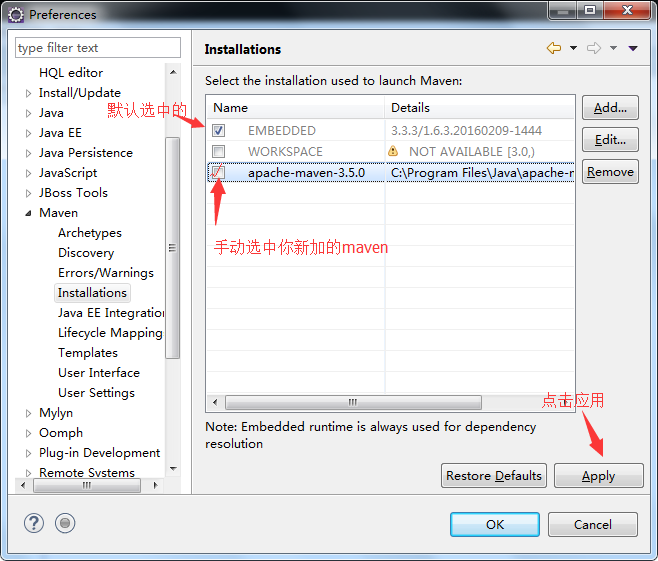




**点击确定之后会出现：**

****

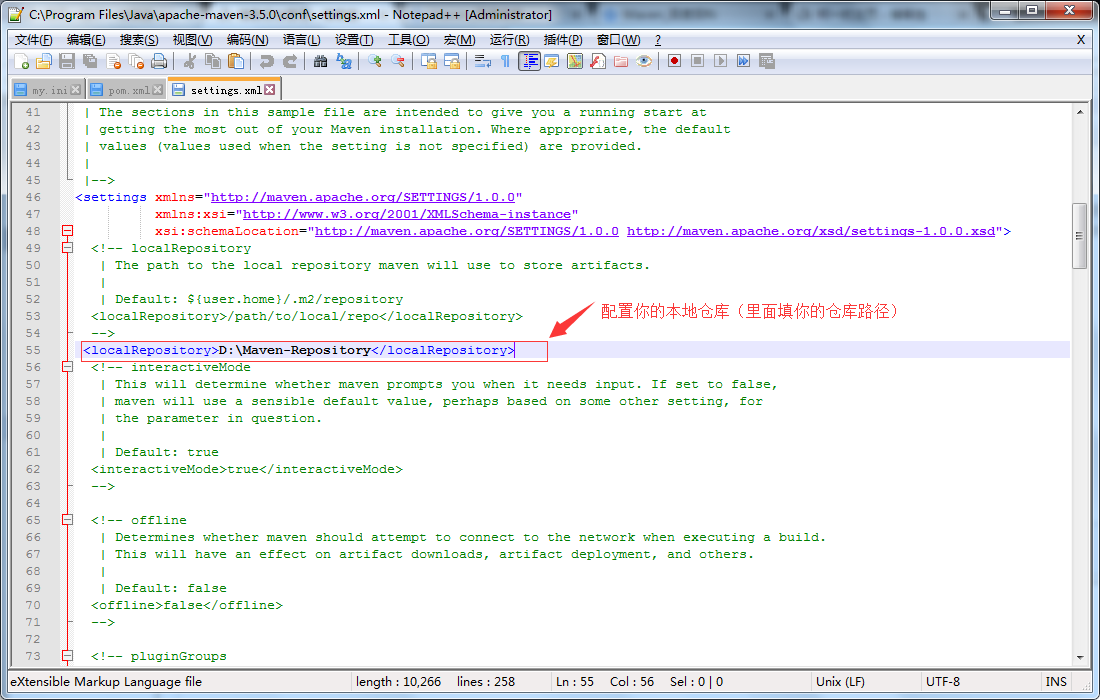
**点击finish之后：**

****

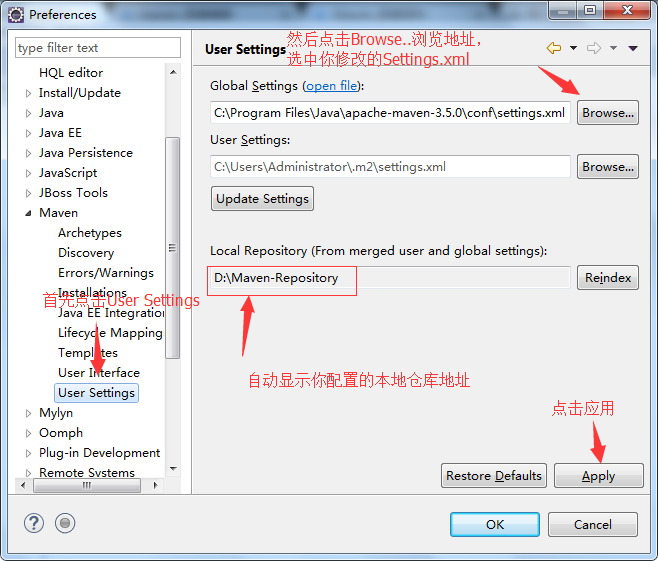
这个时候你就把maven装在你的Eclipse中了，但是我们知道，maven是一个方便我们管理jar包的工具，我们需要用到的jar包都是从maven的中央仓库里下载的，但是我们不需要每次都需要去本地仓库里下载，当我们下载过一次之后就可以在我们的本地仓库中导入jar包，那么，怎么连接本地仓库呢？

### 连接本地仓库

第一步：你首先需要找到你的maven解压文件夹，然后打开conf子文件夹，然后编辑settings.xml



 第二步：回到eclipse中点击window再点击preferences



# CloudCrypto项目

创建Maven项目

## 导入github项目

File > Import > Maven > Existing Maven Project. Point to the location where you cloned the SDK

## 将jars文件夹中的jar包添加到class path中。