中文详细介绍：<http://www.yiibai.com/maven/>

Maven 与 IntelliJ IDEA 的完美结合

你是否正在学习Maven？是否因为Maven难用而又不得不用而苦恼？是否对Eclipse与Maven的冲突而困惑？ 那么我告诉你一个更直接更简单的解决方案: IntelliJ IDEA！

**1. 什么是 Maven，为什么要使用 Maven 而不是 Ant**

Maven简单来说是一个项目管理工具（解决包依赖和项目的自动化构建），被认为是Ant的替代品或者继任者。事实上Maven的功能要远远超出Ant，它不仅仅提供编译的脚本，更是在整个项目周期中提供测试，发布，文档生成等功能，并且有着独特的依赖性管理方法。但是强大的功能的代价就是复杂的使用方法，第一次使用Maven往往需要将 近半个小时的时间（国内网速）来下载一个本地的依赖库，这无疑会使很多开发人员“知难而退”。虽然广受诟病，一个不争的事实就是Maven逐渐代替了 Ant，使用Maven也成了Java开发人员的一个必要技能。

**2. IntelliJ IDEA 与 Eclipse**

在Java的世界里，Eclipse毋庸置疑是IDE中的王者，IntelliJ IDEA和NetBeans都只能生活在Eclipse的光环下。Eclipse的强大已经不用我多说了，但其实IntelliJ IDEA也是一个相当优秀的IDE，尤其体现在它的智能上。它的一个针对.Net的Clone产品ReSharper已经被越来越多的C#开发者所接受。 IntelliJ IDEA提供了大量的智能规则来协助开发者，类似代码补全，用法的转换，格式的提示等等，当你习惯以后，你就会对那个小灯泡爱不释手。

**3. 选择 Maven + IntelliJ IDEA 的理由**

之所以要选择IntelliJ IDEA，是因为Maven+Eclipse实在太难用了。Eclipse有两个Maven插件 m2eclipse 和 Eclipse IAM。但是由于Eclipse和Maven设计上的矛盾，这两个插件可能都达不到你想要的效果。m2eclipse 会把你所有类型的项目都视为Maven类型项目，这对于一个单纯的Java项目或许可以，对于一个JEE或者WEB项目都是一场灾难。Eclipse IAM 并不会改变你的项目类型，但是他提供的功能也就仅仅限于编辑和运行POM文件了，甚至连依赖性都管理不了。相比之下，如果你非要用Eclipse的话我还 是推荐你使用Eclipse IAM。Eclipse想要完美的支持Maven恐怕需要在底层设计上做一些改变，无论是代码规范，项目结构规范和依赖性管理上，都有着不可调和的矛盾。 显然，在这方面IntelliJ IDEA做的要好的多，以至于你很难说出对它的不满。

**4. Maven + IntelliJ IDEA 的优势**

**一切都是Module：**

      IntelliJ IDEA借鉴的Maven的概念，不在采取Eclipse里Project的概念，一切都是Module。无论是否使用Maven，你的项目都是一个独立 的Module。并且你可以混搭使用Maven Module和普通的Java Module，两者可以和谐共存。

**支持Multi-module：**

      可以说Maven的项目结构设计是非常严格的，现实应用中你必须用到Parent-Child的模式。Eclipse由于不支持在一个Project上建 立Sub-Project，因此无论如何目前都不能实现。IntelliJ IDEA可以完美的实现这个设计，并且无论是Module属性里，还是彼此的依赖性上都不会出现问题。

**更方便的生命周期操作：**

      比起Eclipse通通放进右键菜单的行为，IntelliJ IDEA有着单独的窗口可以完成Maven的操作。你可以针对不同Module进行 Clean Compile Package Install等操作，各个Plug-in的操作也一清二楚。

**完美的依赖性管理：**

      由于Maven会把所有依赖的包放在本机的一个目录下，所以实际上是脱离Project本身存在的。IntelliJ IDEA引入了一个External Library的概念，所有的Maven依赖性都会放在这里，和项目自带的库区分开。并且Module之间会智能的判断，你不需要Maven Install来进行引用代码的更新。

**动态更新：**

      每当Maven相关的设置更改时，例如修改了pom的依赖性，添加删除Module，IntelliJ IDEA会提示你进行更新。这种更新实际上就是运行了Maven，所以你不需要手动运行Maven Compile来进行更新，也不会像Eclipse里遇到莫名其妙的问题。

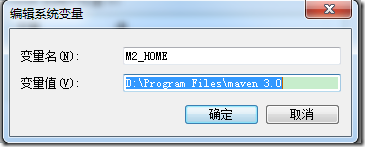
**5. Maven 安装**

**下载：**

到http://maven.apache.org/download.html 下载最新的Maven版本，我下载的版本为最新的apache-maven-3.0.4。推荐使用最新的Maven 3，因为比以前的Maven 2性能更好，而且完全兼容Maven 2。

**配置环境变量：**

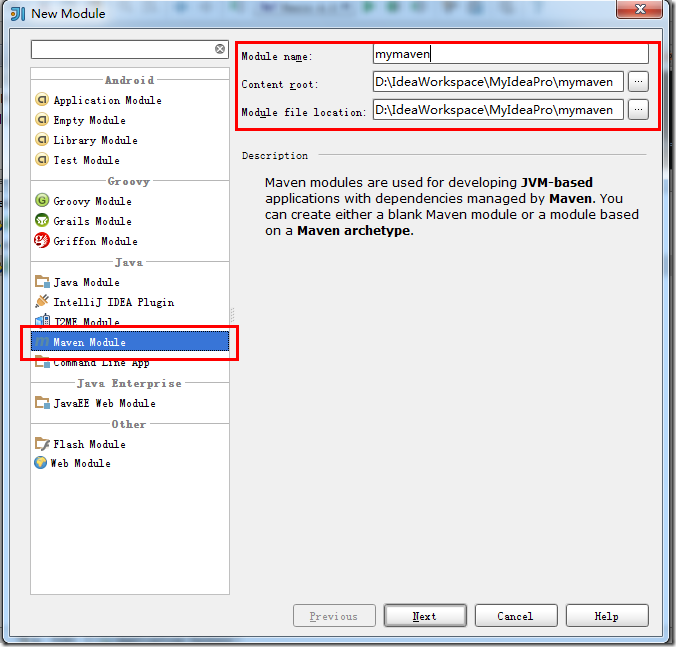
下载好以后，解压到一个目录，我的目录为 D:\apache-maven-3.0.4，接着配置Maven环境变量M2\_HOME为你解压的目录，如下图新建M2\_HOME变量并追加到Path后面：



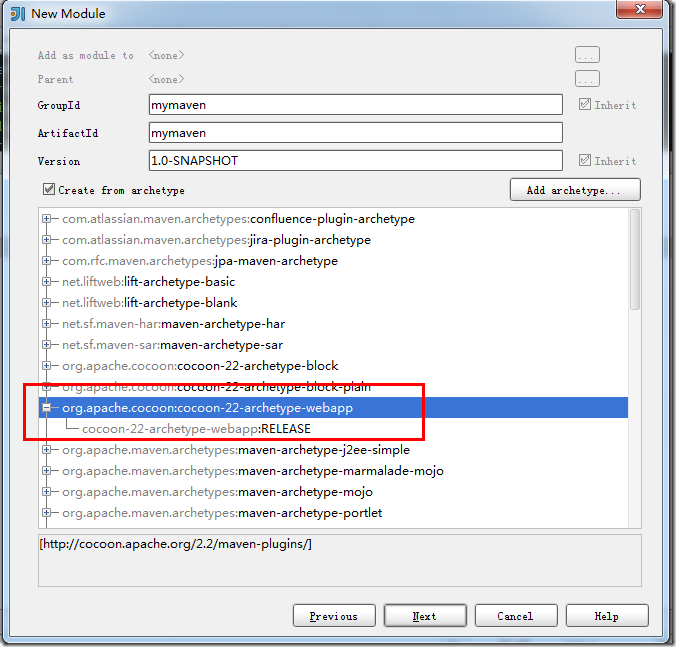
如果要在CMD控制台运行Maven命令的话，还需要配置Jdk环境变量，因为我接下来的操作都是基于IntelliJ IDEA，所以就不用配置JDK环境变量。

**6. Maven 与IntelliJ IDEA完美使用（基于IntelliJ IDEA 12）**

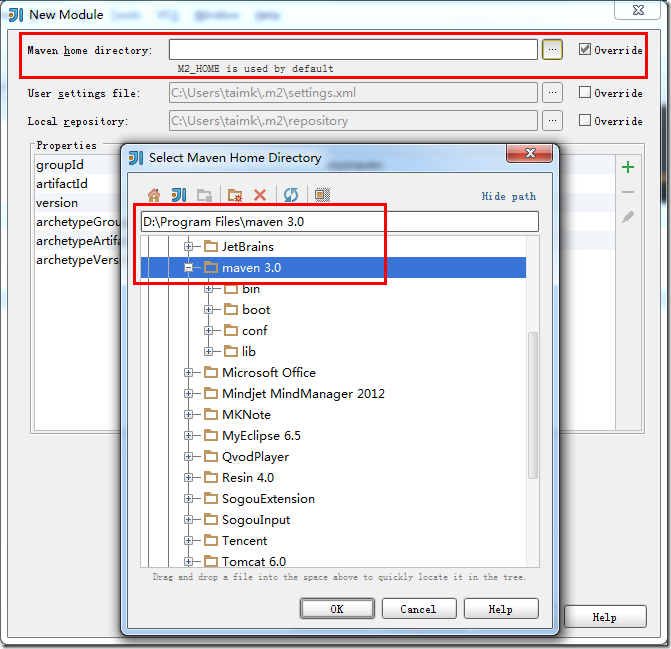
打开IntelliJ IDEA，新建一个Maven项目，如下图，项目名称为mymaven

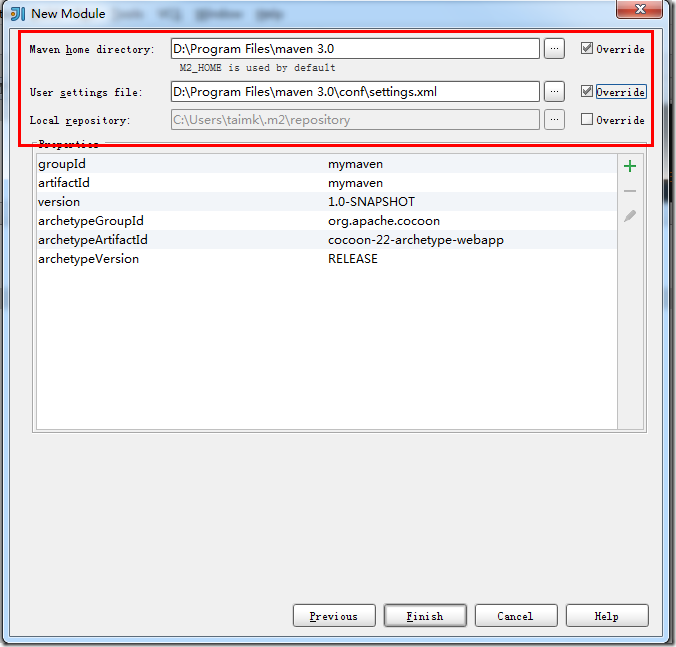


选择Maven模版



创建过程中，配置Maven主目录





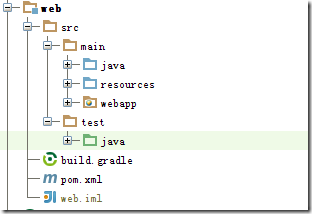
创建好后的目录结构如下图，

IDEA创建了

src/main/resources（自动创建）、

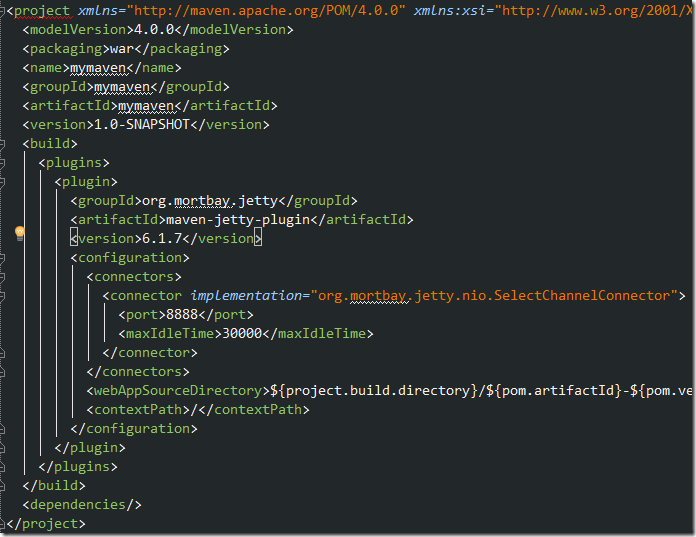
src/main/java（手动创建）、

src/main/test（手动创建）、以及pom.xml文件（自动创建）



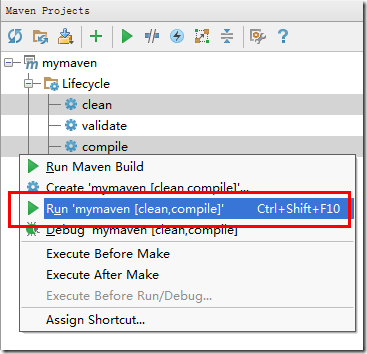
按照Maven约定，项目的主目录位于src/main/resources、src/main/java中，测试目录位于src/main/test 中，所以我们在最终打包项目的时候，只包含主目录代码，不会包含测试目录里的代码，而测试目录里的代码只会在测试时候用到。

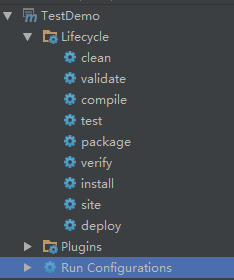
最后pom.xml这个文件是我们Maven项目的核心文件，POM(Project Object Model，项目对象模型)定义了项目的基本信息，用于描述项目如何构建，声明项目依赖。



**project：**是xml文件的根元素，我们主要关心根元素下的子元素   
**modelVersion：**定义了当前POM模型的版本，Maven 2 或Maven 3 只能是4.0.0   
**groupId：**定义了项目属于哪个组，一般来说这个往往和公司或组织关联，比如说，你所在的公司为mycom.那就应该定义为com.mycom.mymaven，mymaven为项目名称   
**artifactId：**定 义了当前Maven项目在组中的唯一id，例如HelloMaven这个项目，我则把他定义为hello-maven，还例如spring项目，他的 groupId为org.springframework，对于content项目来说artifactId则为spring-content，web项 目则为spring-web

最后我们使用IntelliJ IDEA来用Maven编译我们的代码，打开Maven Projects菜单（在view菜单里），如图，选中clean、compile并右键点击Run 'hello-maven[clean,compile]'按钮即可。





由此可见maven可以为项目执行的命令包含clean/validate/compile/test/package/verify/install/site/deploy等

控制台输出依赖和构建信息如下：

D:\java\bin\java "-Dmaven.home=D:\IntelliJ IDEA 2016.2.5\plugins\maven\lib\maven3" "-Dclassworlds.conf=D:\IntelliJ IDEA 2016.2.5\plugins\maven\lib\maven3\bin\m2.conf" -Didea.launcher.port=7532 "-Didea.launcher.bin.path=D:\IntelliJ IDEA 2016.2.5\bin" -Dfile.encoding=UTF-8 -classpath "D:\IntelliJ IDEA 2016.2.5\plugins\maven\lib\maven3\boot\plexus-classworlds-2.4.jar;D:\IntelliJ IDEA 2016.2.5\lib\idea\_rt.jar" com.intellij.rt.execution.application.AppMain org.codehaus.classworlds.Launcher -Didea.version=2016.2.5 clean compile

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building TestDemo 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ TestDemo ---

[INFO]

[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ TestDemo ---

[debug] execute contextualize

[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ TestDemo ---

[INFO] Nothing to compile - all classes are up to date

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 2.613s

[INFO] Finished at: Sat May 06 21:42:32 CST 2017

[INFO] Final Memory: 7M/77M

[INFO] ------------------------------------------------------------------------

根据输出结果我们看到，clean命令让Maven先清理target目录，Maven一般认为项目的输出目录为target，然后执行resource任务，现在未定义项目资源，所以暂时略过，最后执行编译任务！   
  
原文地址：http://my.oschina.net/xianggao/blog/97782?p=2#comments

建立Mian文件，并执行编译compile：

D:\java\bin\java "-Dmaven.home=D:\IntelliJ IDEA 2016.2.5\plugins\maven\lib\maven3" "-Dclassworlds.conf=D:\IntelliJ IDEA 2016.2.5\plugins\maven\lib\maven3\bin\m2.conf" -Didea.launcher.port=7533 "-Didea.launcher.bin.path=D:\IntelliJ IDEA 2016.2.5\bin" -Dfile.encoding=UTF-8 -classpath "D:\IntelliJ IDEA 2016.2.5\plugins\maven\lib\maven3\boot\plexus-classworlds-2.4.jar;D:\IntelliJ IDEA 2016.2.5\lib\idea\_rt.jar" com.intellij.rt.execution.application.AppMain org.codehaus.classworlds.Launcher -Didea.version=2016.2.5 compile

[INFO] Scanning for projects...

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] Building TestDemo 1.0-SNAPSHOT

[INFO] ------------------------------------------------------------------------

[INFO]

[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ TestDemo ---

[debug] execute contextualize

[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO]

[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ TestDemo ---

[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!

[INFO] Compiling 1 source file to F:\Html\mannual\_files\_all\dailynotes\包管理工具\maven\_study\idea\_maven\target\classes

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 2.431s

[INFO] Finished at: Sat May 06 21:46:41 CST 2017

[INFO] Final Memory: 11M/109M

[INFO] ------------------------------------------------------------------------

初级入门：with eclipse

一、前言  
        早就知道maven 在java 项目的管理方面名声显赫，于是就想着学习掌握之，于是查阅了大量文档。发现这些文档的作者都是java 的大腕，大多都是站在掌握了一定maven 基础的角度上进行介绍，让我这初学者看的云里雾里不知所云。于是又去查看maven 的官方网站，总算是有所了解，但一旦动手实际操作却又雾里看花。唉，没办法，就只有一遍一遍的动手尝试，经过种种磨难总算是有一点眉目了。现在就将我的经验写出来，一来避免将来遗忘，二来也给和我一样的菜鸟们提供一点点的方便。呵呵。本文的主题在于实用操作，个中原理还是麻烦大家去查一下java大腕们的文章吧。这里推荐两篇文章：

Maven 2.0:编译、测试、部署、运行  
Maven2 的新特性  
[[url]http://www-128.ibm.com/developerworks/cn/opensource/os-maven2/index.html[/url]](http://www-128.ibm.com/developerworks/cn/opensource/os-maven2/index.html)

二、maven2安装  
1、首先去官方网站下载之：[[url]http://maven.apache.org/download.html[/url]](http://maven.apache.org/download.html)，我选择的是截至目前最新版本maven2.0.4版本  
2、设置环境变量：Maven2下载完后，我将其解压到我的计算机的d:\maven204下，目录结构如下：  
D:\  
|--Maven204  
|-- bin  
|-- conf  
|-- core  
|-- lib  
|-- local

环境变量：（操作系统windows2003）  
我的电脑----属性----高级----环境变量，点击“系统变量”下的新建，输入：变量名 MAVEN\_HOME; 变量值d:\Maven204，在系统变量列表中找到系统变量path，打开之，在变量值中追加”;%MAVEN\_HOME%\bin”，至此环境变量设置完毕。检查一下是否已经完成安装，打开dos窗口，

输入mvn –v，如果出现以下信息则表示maven2已经安装成功：  
X：〉mvn –v  
Maven Version 2.0.4

我已经说得够详细了，还不行？那我只能说你比我还菜。

三、用maven2创建第一个java工程  
1、选择一个根目录，我的java工作目录是D:\eclipse\workspace  
2、打开dos窗口采用cd命令进入D:\eclipse\workspace目录下  
3、输入下列指令  
D:\eclipse\workspace>mvn archetype:create -DgroupId=com.efn -DartifactId=study  
指令执行完毕会出现如下的目录结构：  
study  
|-- pom.xml  
`-- src  
|-- main  
| `-- java  
| `-- com  
| `-- mycompany  
| `-- app  
| `-- App.java  
`-- test  
`-- java  
`-- com  
`-- mycompany  
`-- app  
`-- AppTest.java

四、生成eclipse工程文件  
我采用的开发工具是eclipse,所以用maven的eclipse参数生成eclipse工程文件。输入如下  
1、 采用dos的cd命令进入study目录  
2、 输入指令：mvn eclipse:eclipse 如下所示：  
D:\eclipse\workspace\study>mvn eclipse:eclipse  
指令执行完毕后，发现多了两个文件：.classpath和.project

五、下载maven的eclipse插件  
1、打开eclipse  
2、打开菜单 help -Software updates-Find and Install  
3、选择第二项，进入下一步  
4、点击按钮“New Remote Site…”  
5、在Name中输入：Maven2，在URL中输入[url]http://m2eclipse.codehaus.org/[/url]  
Ok,现在一切都是自动化的，你所需要做的就是等待并按提示按按钮。

六、导入工程  
刚才已经生成了eclipse的工程文件，现在导入之。菜单File-import ，按“下一步”，选择工程文件导入完毕后，工程管理透视图就会出现study节点了，在工程节点上，按鼠标右键，打开弹出菜单，选择maven2菜单项，弹出子菜单->Enable，点击之打开，在Group Id中输入：study即可。

打开study的pom.xml文件，会有如下类似内容：

xml 代码

1. <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 [url]http://maven.apache.org/maven-v4\_0\_0.xsd[/url]">
2. <modelVersion>4.0.0</modelVersion>
3. <groupId>com.efn</groupId>
4. <artifactId>study</artifactId>
5. <packaging>jar</packaging>
6. <version>1.0-SNAPSHOT</version>
7. <name>Maven Quick Start Archetype</name>
8. <url>[url]http://maven.apache.org[/url]</url>
9. </project>

注意：有关这个文件的作用和各节点的作用不是本文的描述重点，可以到maven的官方网站查阅[[url]http://maven.apache.org/[/url]](http://maven.apache.org/)目前为止，所有的准备工作都已经就绪，现在我们就以一个主main函数访问数据库为例介绍在实际开发中的使用。

七、搭建架包  
每一个java工程都不免使用第三方的架包，Maven的好处就是可以统一管理这些架包，并使多个java工程共享这些架包。我们所需要做的就是按规则编辑pom.xml文件。Maven已经非常智能，使用maven的eclipse插件的向导，只要输入参数就可以将配置文件生成出来。我们的例子是用jdbc方式访问oracle的数据库。所以需要jdbc和oracle的架包的支持。下面我们就来搭建这两个架包到study工程里来。

有两种方式可以采用：  
1、自动向导  
第一种是采用向导方式并让向导直接去maven的服务器上下载你需要的架包，这种方式存在一定的风险，一来可能maven的服务器上并没有你需要的东东，二来每次智能的maven总是去寻找那并不存在的东东。抛出一大堆红字提示不说，达不到预期目标才是让人郁闷。不过为了保证文档的完整性，还是给出操作步骤。以junit为例（这个东东倒是没有问题，呵呵）当工程的maven被Enable后，弹出菜单的maven2子菜，选择子菜单的“Add Dependency”菜单项，在Query中输入“junit”，向导会自动列出相关列表供选择。选择你最需要的架包，按“OK” 按钮。  
如果你的本地仓库已经存在该架包，则向导只在pom.xml加入依赖项信息，pom.xml文件的内容如下

xml 代码

1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://[url]www.w3.org/2001/XMLSchema-instance[/url]"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 [[url]http://maven.[/url]](http://maven.apache.org/maven-v4_0_0.xsd)"apache.org/maven-v4\_0\_0.xsd>
2. <modelVersion>4.0.0</modelVersion>
3. <groupId>com.efn</groupId>
4. <artifactId>study</artifactId>
5. <packaging>jar</packaging>
6. <version>1.0-SNAPSHOT</version>
7. <name>Maven Quick Start Archetype</name>
8. <url>[url]http://maven.apache.org[/url]</url>
9. <dependencies>
10. <dependency>
11. <groupId>junit</groupId>
12. <artifactId>junit</artifactId>
13. <version>3.8.1</version>
14. </dependency>
15. </dependencies>
16. </project>

如果本地仓库没有，则向导会去maven的服务器下载架包，并在控制台输出下载日志如下：  
06-8-4 上午09时21分07秒: Local repository folder "" does not exist  
06-8-4 上午09时21分11秒: Reading /study/pom.xml  
06-8-4 上午09时21分11秒: Local repository folder "" does not exist  
06-8-4 上午09时21分18秒: Reading /study/pom.xml  
06-8-4 上午09时21分18秒: Local repository folder "" does not exist  
06-8-4 上午09时21分19秒: Reading /study/pom.xml  
06-8-4 上午09时21分19秒: Local repository folder "" does not exist  
06-8-4 上午09时36分33秒: Local repository folder "" does not exist  
06-8-4 上午09时37分11秒: Reading / study /pom.xml  
06-8-4 上午09时37分11秒: Local repository folder "" does not exist  
06-8-4 上午09时37分15秒: Local repository folder "" does not exist  
06-8-4 上午09时40分07秒: Local repository folder "" does not exist  
06-8-4 上午09时40分08秒: Reading / study /pom.xml  
06-8-4 上午09时40分08秒: Local repository folder "" does not exist  
06-8-4 上午09时46分24秒: Reading / study /pom.xml  
06-8-4 上午09时46分24秒: Local repository folder "" does not exist  
06-8-4 上午09时46分28秒: Local repository folder "" does not exist  
06-8-4 上午09时46分40秒: Local repository folder "" does not exist  
06-8-4 上午09时46分47秒: Local repository folder "" does not exist  
06-8-4 上午09时46分47秒: Reading / study /pom.xml  
06-8-4 上午09时46分47秒: Local repository folder "" does not exist  
06-8-4 上午09 时46 分49 秒: Downloading [central] -> [[url]http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.pom[/url]](http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.pom)  
06-8-4 上午09 时46 分49 秒: Downloaded [central] -> [[url]http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.pom[/url]](http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.pom)  
06-8-4 上午09 时46 分51 秒: Downloading [central] -> [[url]http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.jar[/url]](http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.jar)  
06-8-4 上午09时47分00秒: Downloaded [central] -> [[url]http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.jar[/url]](http://repo1.maven.org/maven2/junit/junit/3.8.1/junit-3.8.1.jar)  
----------------------------------------------------------------  
补充修正：如果出现错误提示同时你手头也有架包的话可以采用maven的指令进行本地化安装。比如我在安装hibernate的架包时告诉我jta无法下载。要求本地化安装，给出的提示如下：  
1) javax.transaction:jta:jar:1.0.1B  
Try downloading the file manually from:[url]http://java.sun.com/products/jta.Then[/url], install it using the command:  
mvn install:install-file -DgroupId=javax.transaction -DartifactId=jta \  -Dversion=1.0.1B -Dpackaging=jar -Dfile=/path/to/filePath to dependency:  
1) com.efn:mywebapp:war:1.0-SNAPSHOT  
2) org.hibernate:hibernate:jar:3.1rc2  
3) javax.transaction:jta:jar:1.0.1B  
----------  
1 required artifact is missing.for artifact: com.efn:mywebapp-1.0-SNAPSHOT.war这个提示是说可以先到sun的网站下载jta架包，然后采用命令行的方式按要求安装，因为我本来就有此架包，于是拷到一个方便的位置，比如c:\,然后在命令行下输入：  
mvn install:install-file -DgroupId=javax.transaction -DartifactId=jta -Dversion=1.0.1B -Dpackaging=jar -Dfile=C:/jta.jar  
执行完毕后，一切OK!  
--------------------------------------------------------------

2、手工配置  
        手工配置比起自动化来说是麻烦了些，不过任何东东掌握在自己手里总归是吃饱喝足谁也不服了不是J。而且配置一次就可以受益终身。更重要的是能解决自动化完成不了的任务。比如我现在要配置oracle的jdbc驱动，采用自动化就提示我下载不了让我改变下载地址等等，真累。  
        算了还是自己来吧。  
        手工配置前还是先介绍一下相关背景知识。首先了解一下jar仓库的概念，maven采用集中化的方式管理架包，凡用maven构建的java工程都可以重用统一管理下的架包，所以必须有一个统一存放jar文件的地方，这就是jar仓库，不论是开发还是部署都会建立一个本地仓库。这个仓库的位置默认是X:\Documents and Settings\Administrator.LIPENG\.m2\repository（X是你操作系统所在盘符）下，你可以修改配置文件改变默认的输出位置。该配置文件在maven目录下的conf子目录下。文件名叫settings.xml。在配置文件中加入如下节点信息即可D:/Repository。不过不建议改变，好像eclipse的maven插件只认默认的输出位置。我就是在这方面出了问题搞了半天才发现是输出路径的问题，也许是我还  
没有玩转插件。谁要是解决此问题别忘了跟大家共享。  
现在我们分析一下配置文件pom.xml的结构，以junit为例，格式如下：

xml 代码

1. <dependencies>
2. <dependency>
3. <groupId>junit</groupId>
4. <artifactId>junit</artifactId>
5. <version>3.8.1</version>
6. </dependency>
7. </dependencies>

在来看看本地仓库目录结构  
Repository  
`-- junit  
|-- junit  
| `-- 3.8.1  
| `-- junit-3.8.1.jar  
现在大家应该明白了吧，多余的话不说啦。照葫芦画瓢就是。不过注意先建目录后写配置文件，否则一旦保存，智能的插件就马上开始下载了…

现在开始手动建立oracle的jdbc目录并配置文件,首先建立目录结构如下：  
Repository  
`-- ojdbc  
|-- ojdbc  
| `-- 14  
| `-- ojdbc-14.jar  
如果你手头的jar文件名叫ojdbc14.jar，则改为ojdbc-14.jar,写配置文件:

xml 代码

1. <dependency>
2. <groupId>ojdbc</groupId>
3. <artifactId>ojdbc</artifactId>
4. <version>14</version>
5. </dependency>

那么现在一个完整的pom.xml文件如下：

xml 代码

1. <?xml version="1.0"?>
2. <project>
3. <modelVersion>4.0.0</modelVersion>
4. <groupId>com.mycompany.app</groupId>
5. <artifactId>myapp</artifactId>
6. <name>Maven Quick Start Archetype</name>
7. <version>1.0-SNAPSHOT</version>
8. <url>[url]http://maven.apache.org[/url]</url>
9. <dependencies>
10. <dependency>
11. <groupId>ojdbc</groupId>
12. <artifactId>ojdbc</artifactId>
13. <version>14</version>
14. </dependency>
15. <dependency>
16. <groupId>junit</groupId>
17. <artifactId>junit</artifactId>
18. <version>3.8.1</version>
19. </dependency>
20. </dependencies>
21. </project>

保存之，则发现工程管理透视图发生了一点变化,依此方法再加上jdbc的架包，现在可以开始写程序了，建一个类并添加main函数，编写程序如下：

java 代码

1. public static void main( String[] args )
2. {
3. Connection conn = null;
4. PreparedStatement ps = null;
5. ResultSet rs = null;
6. try {
7. Class.forName("oracle.jdbc.driver.OracleDriver");
8. conn = DriverManager.getConnection("jdbc:oracle:thin:@(description=(address\_list=(address=(protocol=TCP)(port=1521)(host=192.168.0.240)))(connect\_data=(SERVER = DEDICATED)(SERVICE\_NAME = db.efriendnet.com)))", "efnx", "efnx");
9. ps = conn.prepareStatement("select \* From tb\_partyinfo");
10. rs = ps.executeQuery();
11. while(rs.next())
12. {
13. System.out.println(rs.getString("topic"));
14. }
15. } catch (Exception e) {
16. System.out.print(e.getMessage());
17. }
18. finally
19. {
20. if (rs != null) {try {rs.close();} catch (SQLException e) {}}
21. if (ps != null) {try {ps.close();} catch (SQLException e) {}}
22. if (conn != null) {try {conn.close();} catch (SQLException e) {}}
23. }
24. }

别忘了import相应的包

八、编译程序  
采用maven构建系统，则编译过程就独立了出来。这时你再用eclipse自带的编译工具就不起作用了。所以要想编译、调试、运行还要做一些工作。以前是在dos命令行方式下进行编译，现在的插件很好用，在eclipse配置一下就可以编译了。很方便。现在就做一个介绍。

Eclipse有一个扩展工具就是用来集成其他构建工具的在工程的节点上点击鼠标右键，选择属性，在“编译”的右边窗口选择”新建”按钮，在对话框的“name”中输入：study,点击“Browse Workspace…”列出工程列表供选择。  
选择完毕后，在goals中输入package。别忘了apply.好了，让我们Run吧。如果一切正常，  
控制台会打出maven的编译信息如下：  
[INFO] ---------------------------------------------------------------------  
[INFO] Building Maven Quick Start Archetype  
[INFO] task-segment: [package]  
[INFO] ---------------------------------------------------------------------  
[INFO] resources:resources  
[INFO] Using default encoding to copy filtered resources.  
[INFO] compiler:compile  
[INFO] Nothing to compile - all classes are up to date  
[INFO] resources:testResources  
[INFO] Using default encoding to copy filtered resources.  
[INFO] compiler:testCompile  
[INFO] Nothing to compile - all classes are up to date  
[INFO] surefire:test  
[INFO] Surefire report directory: D:\eclipse\workspace\study\target\s  
urefire-reports  
-------------------------------------------------------  
T E S T S  
-------------------------------------------------------  
Running com.efn.AppTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.157 sec  
Results :  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
[INFO] jar:jar  
[INFO] Building jar: D:\eclipse\workspace\study\target\study-1.0-SNAPSHOT.jar  
[INFO] ----------------------------------------------------------------------------  
[INFO] BUILD SUCCESSFUL  
[INFO] ---------------------------------------------------------------------  
[INFO] Total time: 4 second  
[INFO] Finished at: Fri Aug 04 10:55:42 CST 2006  
[INFO] Memory 2M/7M  
[INFO] --------------------------------------------------------------------  
注意，别忘了每一次程序改动完毕后都要经过这一步编译。因为这是maven的编译器！

九、调试程序  
经过以上步骤我们已经完成了mave管理下的软件生命周期，但是作为一个程序开发人员我们还要进行调试。这里的调试设置和普通的java程序的调试是一样的。  
首先，打开debug对话框：  
        因为是一个一般的java应用程序，所以我们选择 Java Application,点击“New”按钮，输入一些相应的参数，apply-Debug  
Ok,一切正常！希望你也顺利！

十、结束语  
本文只是简单的对maven的操作步骤做一个指南性的说明，实际应用还有很多东西需要实践。如果发现哪里有错误和纰漏也欢迎批评指正！

参考：<http://fluagen.blog.51cto.com/146595/40086/>