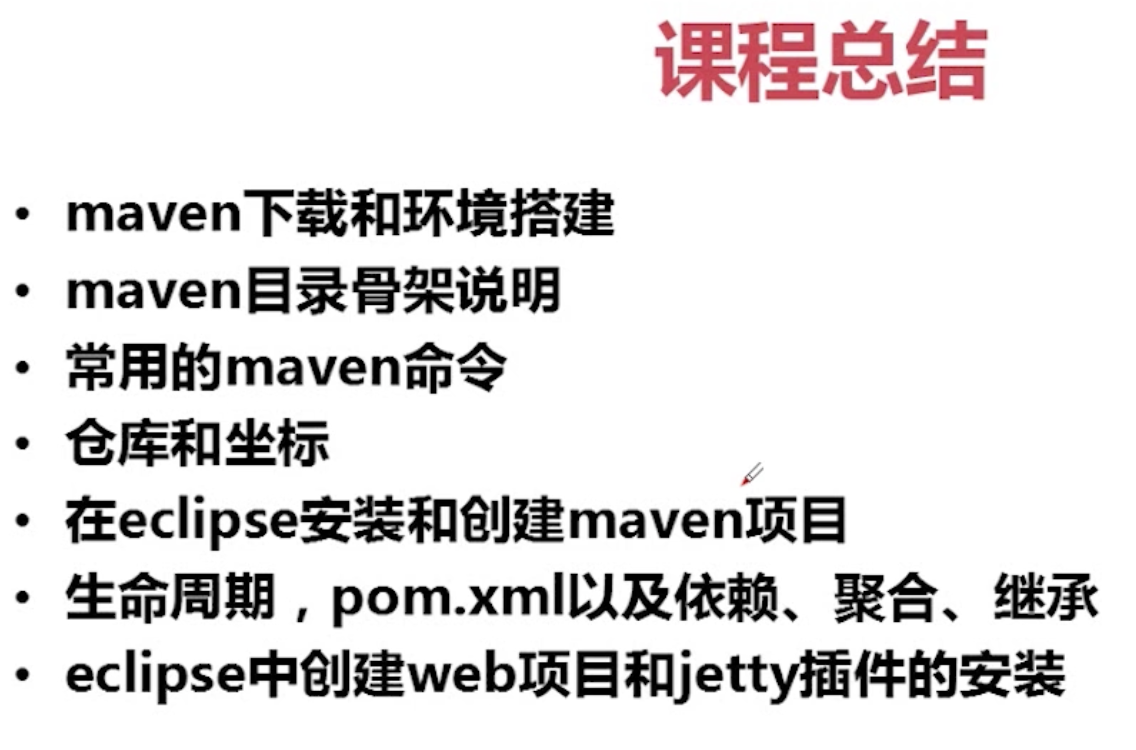


**Maven使用的最终的总结**



1. **maven下载**在apache官网 <https://maven.apache.org/download.cgi>
2. **环境搭建**：
   1. maven环境变量的配置(Mac端，Windows自行百度)：

打开环境变量配置文件（过程类似java的环境变量配置）

打开终端

cd 到根路径， cd ~

vi .bash\_profile （新建配置文件,如果配置过其它环境变量如java等，则不用新建此文件）

以文本方式修改此文件 open -e .bash\_profile

键入以下配置并保存

MAVEN\_HOME=/Library/apache-maven-3.5.3

PATH=$PATH:$MAVEN\_HOME/bin

export MAVEN\_HOME

export PATH

回回到终端使文件生效 source .bash\_profile

打开终端mvn -v查看是否安装成功

**环境变量配置完成后配置settings.xml文件**

* 1. eclipse配置：

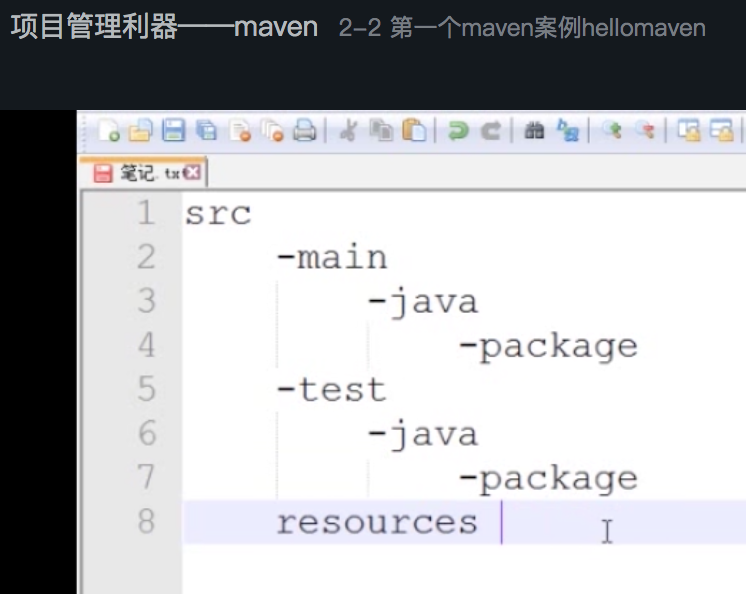
高版本不用安装插件（oxygen），本身自带。选择一下installation使用本地的maven就可以

配置一下路径：使用哪个settings.xml文件的，最好和本地maven使用的settings.xml区别开，复制一个在本地仓库中localRepository

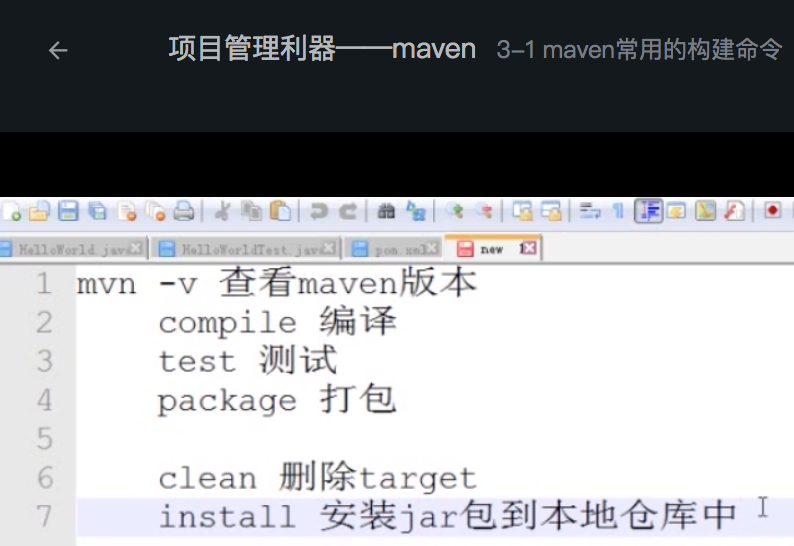
配置一下路径：本地资源库localRepository 位置

3 maven骨架目录

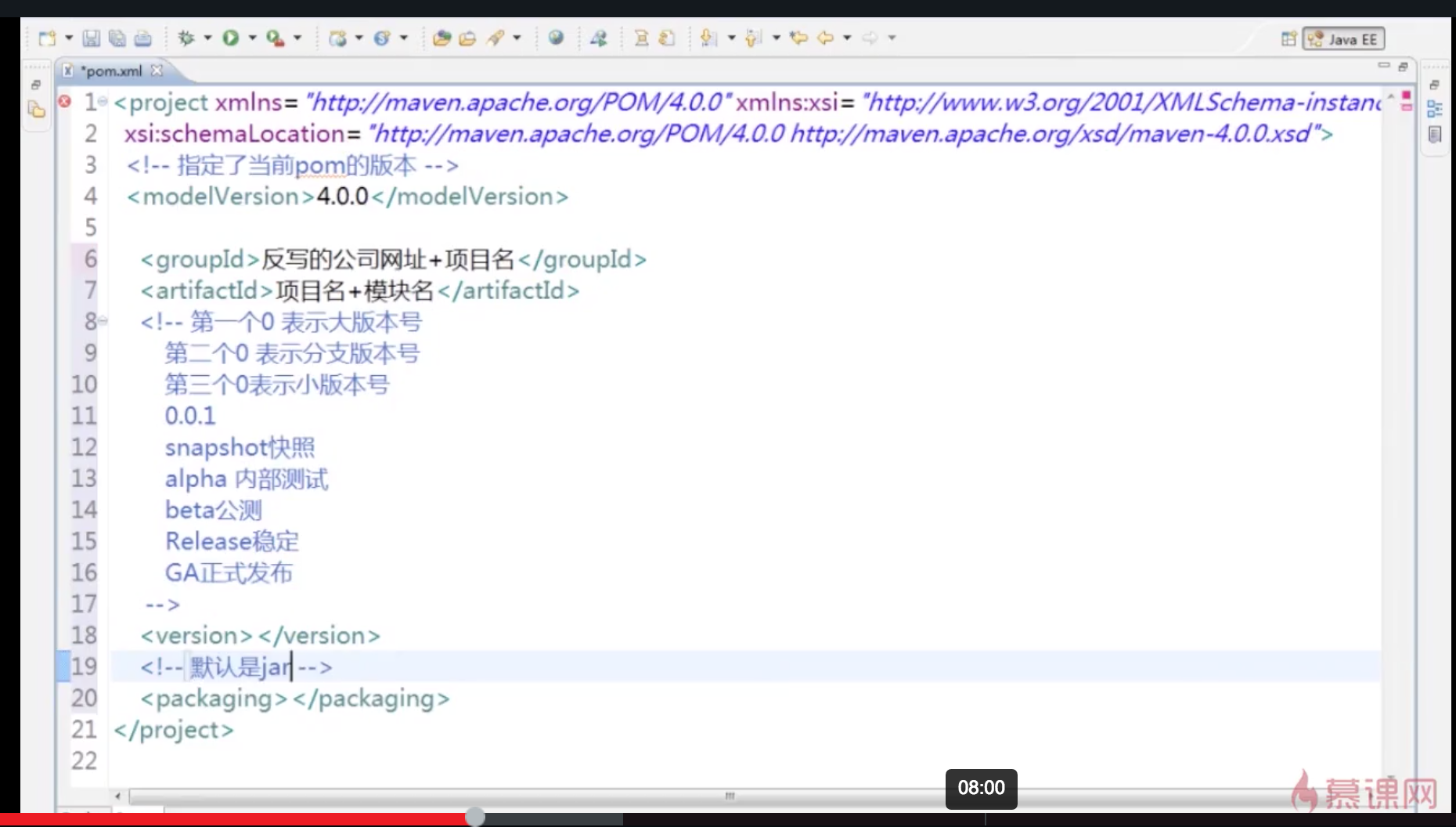
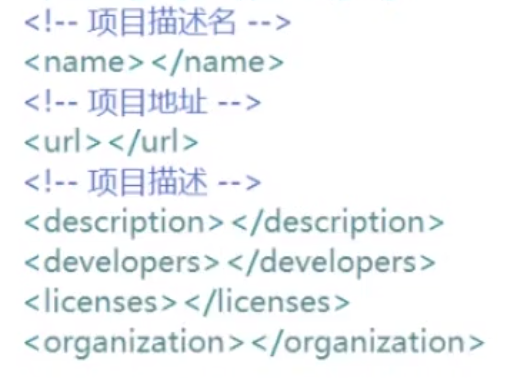




4 常用的maven命令



5 pom.xml



6 maven坐标

引博客：<https://www.cnblogs.com/AlanLee/p/6156268.html>

Maven的一个核心的作用就是管理项目的依赖，引入我们所需的各种jar包等。为了能自动化的解析任何一个Java构件，Maven必须将这些Jar包或者其他资源进行唯一标识，这是管理项目的依赖的基础，也就是我们要说的坐标。包括我们自己开发的项目，也是要通过坐标进行唯一标识的，这样才能才其它项目中进行依赖引用。

**1.何为Maven坐标**

关于坐标（Coordinate），大家最熟悉的应该就是数学中的坐标了吧，我印象不是特别深刻，说的数学专业一点叫作平面几何。在一个平面坐标系中，有x轴和y轴，x轴就是横线，y轴就是竖线，坐标(x,y)表示这个点距离x轴为y，距离y轴为x的一点，任何一个坐标都能唯一标识该平面中的一点。

在实际的生活中，我们可以将地址看成是一种坐标。不同的省，不同的市，不同的区，不同的街道等一系列信息标识了每一个不同的地址。在深圳经常吃外卖的人应该是有体会的，送外卖的小哥都会根据你填写的地址将外卖送给你，而你填的地址就标识了唯一的一个地址。

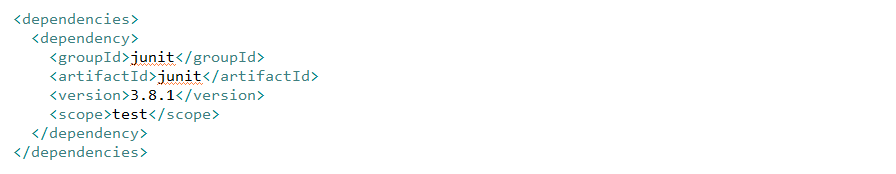
坐标就好比每一个Java构件的身份证一样。Maven的世界是拥有非常多的Java构件的，可能是jar、可能是war、也可能是其他的一些东西。假如Maven中没有坐标的概念，我们是无法来区分这些构件的，所以我们要唯一标识每一个构件。不然就和传统的手工方式一样，你需要spring就去spring下载spring的包，需要mysql又去mysql下载mysql的包，没有统一的规范又怎么能够自动化的去依赖这些构件。

Maven便给我们制定了一套规则那就是使用坐标进行唯一标识。Maven的坐标元素包括groupId、artifactId、version、packaging、classfier。只要我们提供正确的坐标元素，Maven就能找到对应的构件，首先去你的本地仓库查找，没有的话再去远程仓库下载。如果没有配置远程仓库，会默认从中央仓库地址(http://repo1.maven.org/maven2)下载构件，该中央仓库包含了世界上大部分流行的开源项目构件，但不一定所有构件都有，我在以前的开发中就遇到过找不到oracle数据库jar包的问题，不知道现在是否依然如此。

在我们自己开发项目的时候，也是要给我们的项目定义坐标的，这是强制性要求，只有这样，其他项目才能引用该项目的构件。

**2.坐标详细说明**

Maven坐标是通过groupId、artifactId、version、packaging、classfier这些元素来定义的，我们来看看示例：



我们在平时的开发中一般只需要使用必要的几个元素就好了，不必要的一些元素往往是用不着的，我们来看看每个元素所代表的含义，前面也是有简单的提到过的。

**groupId** ：定义当前Maven项目隶属的实际项目。首先，Maven项目和实际项目不一定是一对一的关系。比如SpringFrameWork这一实际项目，其对应的Maven项目会有很多，如spring-core,spring-context等。这是由于Maven中模块的概念，因此，一个实际项目往往会被划分成很多模块。其次，groupId不应该对应项目隶属的组织或公司。原因很简单，一个组织下会有很多实际项目，如果groupId只定义到组织级别，而后面我们会看到，artifactId只能定义Maven项目（模块），那么实际项目这个层次将难以定义。最后，groupId的表示方式与Java包名的表达方式类似，通常与域名反向一一对应。上例中，groupId为junit，是不是感觉很特殊，这样也是可以的，因为全世界就这么个junit，它也没有很多分支。

**artifactId**: 该元素定义当前实际项目中的一个Maven项目（模块），推荐的做法是使用实际项目名称作为artifactId的前缀。比如上例中的junit，junit就是实际的项目名称，方便而且直观。在默认情况下，maven生成的构件，会以artifactId作为文件头，如junit-3.8.1.jar，使用实际项目名称作为前缀，就能方便的从本地仓库找到某个项目的构件。

**version** : 该元素定义了使用构件的版本，如上例中junit的版本是3.8.1，你也可以改为4.0表示使用4.0版本的junit。

**packaging** ：定义Maven项目打包的方式，使用构件的什么包。首先，打包方式通常与所生成构件的文件扩展名对应，如上例中没有packaging，则默认为jar包，最终的文件名为junit-3.8.1.jar。也可以打包成war等。

**classifier**: 该元素用来帮助定义构建输出的一些附件。附属构件与主构件对应，如上例中的主构件为junit-3.8.1.jar,该项目可能还会通过一些插件生成如junit-3.8.1-javadoc.jar,junit-3.8.1-sources.jar, 这样附属构件也就拥有了自己唯一的坐标。

上述5个元素中，groupId、artifactId、version是必须定义的，packaging是可选的（默认为jar），而classfier是不能直接定义的，需要结合插件使用。

大家可以发现一个构件的名称就是由这几个元素的值所组成的，junit-3.8.1.jar，自己去比对一下，不仅如此，Maven仓库的布局也是基于Maven坐标的，大家可以看看本地仓库的布局是否如此。

理解了Maven坐标以后，大家就能够开始去学习Maven的依赖管理了。