**Maven编码规范**

**<V1.0>**

编 写：

审 核：

批 准：

# 前言

## 传统项目开发过程

在进行软件开发的过程中，无论什么样的项目，采用何种技术，使用何种编程语言，我们都要重复相同的开发步骤：收集依赖构建、编写源代码、配置信息、测试、打包、发布、文档

实际上这些步骤是完全重复性的工作。而maven要做的就是让开发人员能专注于商业逻辑并去实现它，而不是把时间浪费在学习如何在不同的环境中去打包，发布等等事情。它让开发人员从这些任务中解脱出来，是的项目开发的效率更高。

## Maven优势

maven是面向技术层面，针对Java开发项目管理工具，它提供了构建工具所提供功能的超集，除了构建功能之外，Maven还可以管理项目结构、管理依赖关系、生成报告、生成Web站点、同时还有助于团队成员之间的交流与协作。开发者通过Maven管理项目从中受益，Maven为开发提供以下便利之处

* 指导开发

提供了Java项目的最佳开发实践，自由开发项目骨架而可以自动生成项目结构

* 自动编译

不仅仅只像Ant自动编译，还包括测试、打包、发布、文档生成、项目站点生成等等。

* 依赖管理

Maven可以方便地管理应用程序依赖，例如第三方依赖、模型依赖等

* 无限扩展

插件模式可以无限增强Maven功能，例如通过Tomcat、Jetty插件可以自由控制其服务器。

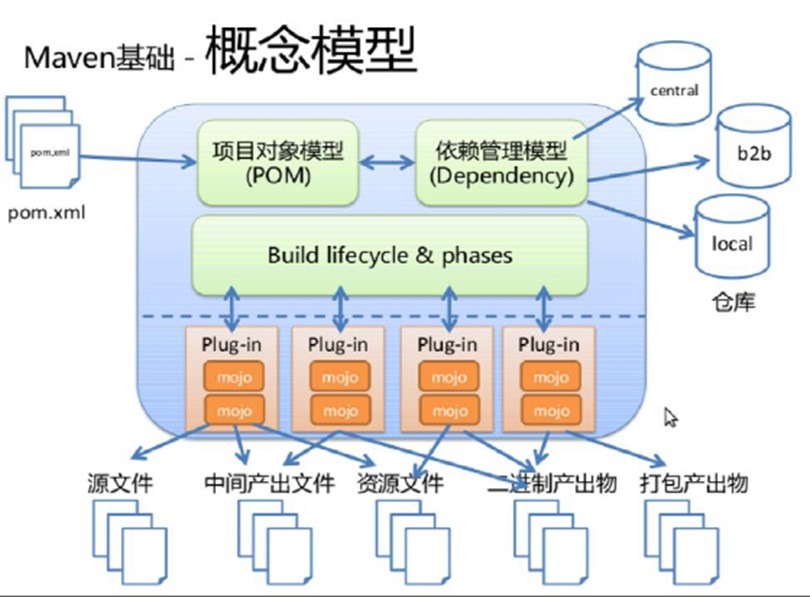
* 持续集成

鼓励开发者积极提交代码，更早地发现程序错误，在并行开发中稳妥推进。

* 开发协作

更简单和谐的团队协作

## Maven概念模型



# 基本概念

Maven主要服务基于java平台的项目构建、依赖管理和项目信息管理

* 项目对象模型（Project Object Model）
* 坐标（Coordinates）
* 项目生命周期（ProjectLifecycle）
* 插件（plugin）和目标（goal）
* 依赖管理系统（Dependency Management System）
* 仓库管理（Repositories）

官网：<http://maven.apache.org>

## Maven名词解释

* Project

任何你想build的事物，Maven都可以认为它们是工程。这些工程被定义为工程对象模型（POM，Project Object Model）。一个工程可以依赖其它的工程，一个工程也可以由多个子工程构成。

* POM

POM（pom.xml）是Maven的核心文件，它是指示Maven如何工作的元数据文件，类似于Ant中的build.xml文件。POM文件位于每个工程的根目录中。

* GroupId

groupId是一个工程在全局中唯一的标识符，一般的，它就是工程名。groupId有利于使用一个完全的包名，将一个工程从其它有类似名称的工程里区别出来。

* Artifact

artifact是工程将要产生或需要使用的文件，它可以使jar文件，源文件，二进制文件，war文件，甚至是pom文件。每个artifact都由groupId和artifactId组合的标识符唯一识别。需要被使用（依赖）的artifact都要放在仓库（Repository）中，否则Maven无法找到它们。

* Dependency

为了能够build或运行，一个典型的Java工程会依赖其它的包。在Maven中，这些被依赖的包就被称为dependency。dependency一般是其它工程的artifact。

* Plug-in

Maven是由插件组织的，它的每一个功能都是由插件提供的。插件提供goal，并根据在POM中找到的元数据去完成工作。

* Repository

仓库

## setting.xml

* $user.home/.m2/repository/setting.xml

为用户范围的配置文件。

* $M2\_HOME/conf/setting.xml

为全局范围的配置文件，修改后将影响本机所有用户的配置。这里建议只修改用户级别的配置，既不影响其他用户，也不影响后期升级。

* 配置介绍

**localRepository**：自定义本地库路径，默认在$user.home/.m2中。

**interactiveMode**：Maven是否需要和用户交互以获得输入。默认为true

**offline**：表示Maven是否需要在离线模式下运行。如果构建系统需要在离线模式下运行，则为true，默认为false。当由于网路设置原因或者安全因素，构建服务器不能连接远程仓库的时候，该配置就十分有用。

**pluginGroups**：当插件的组织Id（groupId）没有显示提供时，供搜寻插件组织Id的列表。该元素包含一个pluginGroup元素列表，每个子元素包含了一个组织Id。当我们使用某个插件，并且没有再命令行为其提供组织Id的时候，Maven就会使用该列表。默认情况下该列表包含了org.apache.maven.plugins

**proxies**：通过代理访问外部库

**servers**：集成认证服务，例如集成Tomcat

**mirrors**：为仓库列表配置的下载镜像列表

**profiles**：个性配置，需要在Activation标签中激活

**activeProfiles**：表示激活的profile

更过关于setting.xml配置的介绍请参见：http://www.cnblogs.com/skyme/archive/2011/08/19/2146105.html

## Maven仓库

* 远程公用仓库

Maven内置了远程公用仓库：http://repo1.maven.org/maven2

这个公共仓库是由Maven自己维护，里面有大量的常用类库，并包含了世界上大部分流行的开源项目构件。目前是以Java为主的。

* 内部中心仓库

内部中心仓库也称为私有共享仓库（私服）。一般是由公司自己设立的，只为本公司内部共享使用。它既可以作为公司内部构件协作和存档，也可以作为公用类库镜像缓存，减少在外部访问和下载的频率。

搭载私服的方式有多重，这里就不再阐述。（公司私服地址：http://nexus.bsdn.org/content/groups/public/ ）

* 本地仓库

Maven会将工程中依赖的构件（jar包）从远程下载到本机一个目录下管理，通常默认在$user.home/.m2/repository下。

自Maven2以后，构件的存储方式通常是groupId/artifactId/version/\*.jar。除了以上默认的本地库地址，我们也可以自定义本地库到指定路径中去。

* 修改本地库位置

在$M2\_HOME/conf/setting.xml文件的<localRepository>元素中指定路径，例如：<localRepository>D:/my\_repository</localRepository>

## Maven常用指令

* 检测Maven、JDK版本

mvn -v 或者 mvn -version

* 获取帮助选项

mvn -h 或者 mvn -help

* 显示详细错误信息

mvn -e

* 创建Java项目

mvn archetype:create

-DgroupId=${groupId} -DartifactId=${artifactId}

例如：

mvn archetype:create

-DgroupId=com.bstek -DartifactId=hello -Dversion=0.1

* 创建Web项目

mvn archetype:create

-DgroupId=${packageName}

-DartifactId=${webappName} -DarchetypeArtifactId=maven-archetype-webapp

* 创建其它项目（例如SSH）

mvn archetype:generate

然后根据提示选择项目骨架、groupId、artifactid、版本号••

* 转换成Eclipse工程

mvn eclipse:eclipse

mvn eclipse:clean //清楚Eclipse设置信息

* 编译

mvn complie

* 编译测试代码

mvn test-compile

* 产生Site

mvn site

* 测试

mvn test //运行测试

mvn test -Dtest=${类型} //单独运行测试类

* 清除

mvn -clean //将清除原来编译的结果

* 打包

mvn package

mvn package -Dmaven.test.skip=true //打包时不执行测试

* 发布

mvn install //将项目打包成构件安装到本地仓库

mvn deploy //发布到本地仓库或服务器

* 手动添加构件到仓库

mvn install:install-file -Dfile=${jar包文件位置} -DgroupId=${groupId} -DartifactId=${artifactId} -Dversion=${版本号} -Dpackaging=jar -DgeneratePom=${是否同时创建pom文件}

* 复制依赖构件到相应目录

mvn dependency:copy-dependencies -DoutputDirectory=${目标目录} -DexcludeScope=${scope} -Dsilent=true

示例：

mvn

dependency:copy-dependencies -DoutputDirectory=WebRoot/WEB-INF/lib

-Dsilent=true -DincludeScope=runtime

* 显示一个插件的详细信息

mvn help:describe -Dplugin=pluginName -Ddetail

## pom.xml

是Maven项目的核心配置文件，位于每个工程的根目录，指示Maven工作的元数据文件。

* 节点介绍

<project > ：文件的根节点

<modelversion > ： pom.xml使用的对象模型版本

<groupId > ：创建项目的组织或团体的唯一 Id

<artifactId > ：项目的唯一 Id, 可视为项目名

<packaging > ：打包类型，一般有JAR,WAR,EAR 等

<version > ：产品的版本号

<name > ：项目的显示名，常用于 Maven 生成的文档。

<url > ：组织的站点，常用于 Maven 生成的文档

<description > ：项目描述，常用于 Maven 生成的文档。

<dependencies>：构件依赖

<parent>：模型继承

<dependencyManagement>：依赖管理

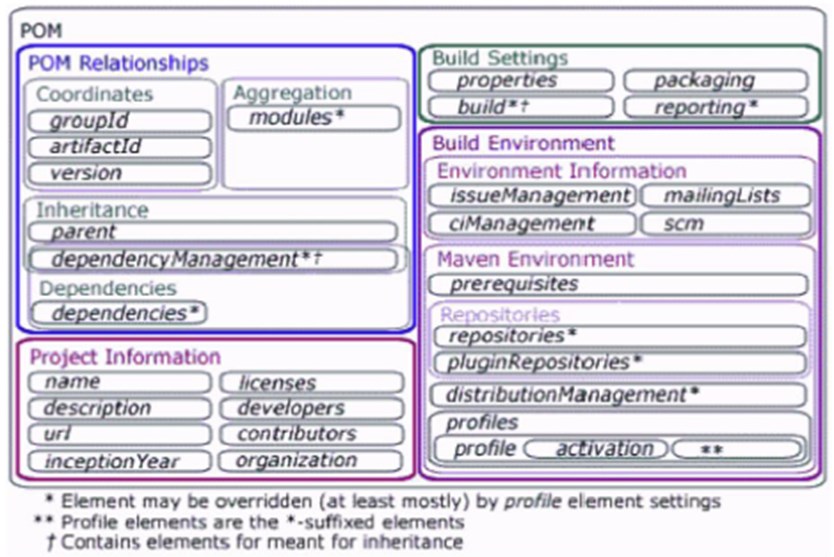
<reporting>：创建报告

<build>：构建

<repositories>：引用第三方仓库

<licenses>：许可

* POM全景图



## 坐标

Maven拥有数量非常巨大的构件，也就是我们平时用的一些jar、wa等文件。Maven定义了这样一组规则：世界上任何一个构件都可以使用maven坐标唯一标识。Maven的坐标元素包括groupId、artifactId、version、packaging、classifier。

groupId：定义当前maven项目隶属于的实际项目（组织机构）。groupId的表示方式与java包名的表示方式类似，通常与域名反向一一对应。

artifactId：该元素定义实际项目中的一个maven项目/模块。

Version：版本

Package：打包方式。如：jar、war。

Classifier：不能直接定义，用来表示构件到底用于何种jdk版本。

## 依赖（dependency）

java工程依赖的其它包。被依赖的包就成为dependency.

Dependency一般是其它工程的坐标。依赖具有传递性。

## 生命周期

Maven的生命周期就是为了所有的构建过程进行抽象和统一。这个生命周期包含了项目的清理、初始化、编译、测试、打包、集成测试、验证、部署和站点生成等几乎所有构建步骤。

Maven拥有三套相互独立的生命周期，他们分别为clean、defaulet和site。

Clean生命周期的目的是清理项目，default生命周期的目的是构建项目，site生命周期的目的是建立项目站点。

## 阶段（phase）

每个生命周期包含一些阶段，这些阶段是有顺序的，并且后面的阶段依赖于前面的阶段，用户和maven最直接的交互方式就是调用这些生命周期阶段。

较之于生命周期阶段的前后依赖关系，三套生命周期是相互独立的，用户可以仅仅调用clean生命周期的某个阶段，或者仅仅调用default生命周期的某个阶段，而不会对其它生命周期产生任何影响。

## 插件及其目标

Maven的核心仅仅定义了抽象的生命周期，具体的任务是交由插件完成的，插件以独立的构建形式存在。

对于插件本身，为了能够复用代码，它往往能够完成多个任务。

例如：maven-dependency-plugin插件，能够基于项目依赖做很多事情。比如，能够分析项目依赖，找到无用的或者重复的依赖；能够列出项目的依赖树。这些功能往往背后有很多可以复用的代码，因此，可以把这些功能聚集在一个插件里，每个功能就是一个插件目标。

调用插件目标方法：例 mvn dependency:tree.冒号前面是插件的前缀，冒号后面是该插件的目标。

# 安装

## Maven安装

* 下载安装包

    地址：<http://maven.apache.org/download.html>

    当前版本为3.0.4

* windows系统下安装

    1）首先需要确定系统中装有JDK（版本1.4+）

    2）将压缩包解压到本地磁盘某目录中，该目录则称为安装目录。

            例如 E:\bin\apache-maven-3.0.4

    3）目录结构介绍：

            bin：Maven的运行脚本

            boot：Maven自己的类装载器

            conf：该目录下包含了全局行为定制文件setting.xml

            lib：Maven运行时所需的类库

* 配置环境变量

    1）M2\_HOME=安装目录



    2）在path变量中增加%M2\_HOME%\bin



    3）MAVEN\_OPTS=-Xms NNNm -Xmx NNNm（非必要项，可防止内存溢出，其中NNN表示具体的内存数量）

* 检查安装正确性

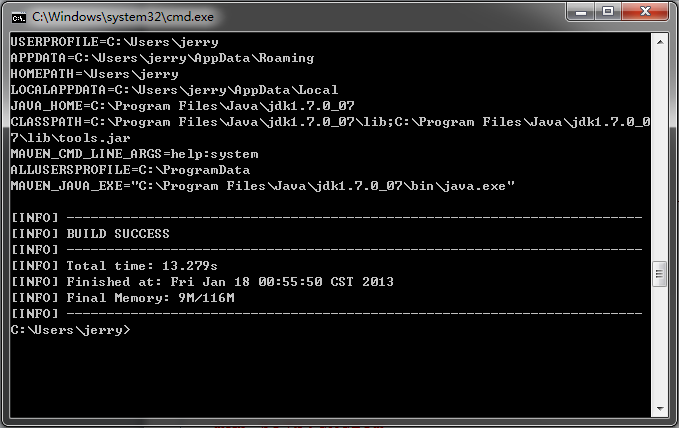
在命令行提示符下执行maven命令：mvn -v



* 实验Maven命令

在命令提示符下面输入：mvn help:system

该命令将会下载help插件并运行它，且打印Java系统属性和环境变量。



## Maven插件安装

为Eclipse提供JDK支持

Eclipse是运行在JRE之上，但Maven需要JDK的一些支持，需要指定JDK，在Eclipse安装目录中的eclipse.ini文件中增加：

-vm

${%JAVA\_HOME%\bin\javaw.exe}

安装M2Eclipse插件

打开Eclipse，找到以下功能界面：

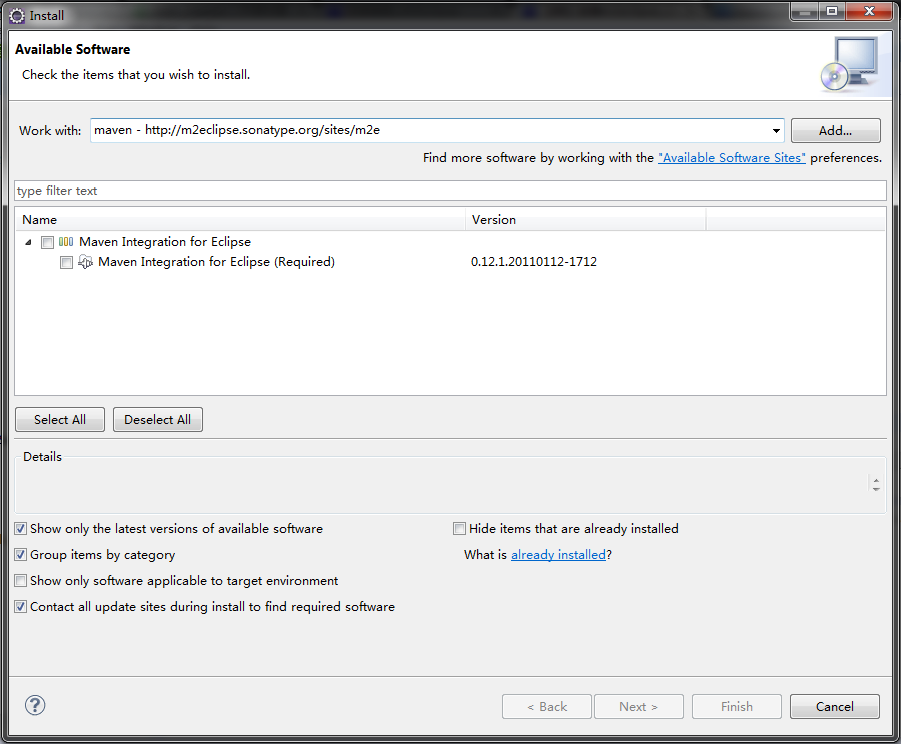
Help -> InstallNewSoftware -> Work with -> Add

分别加入m2的核心组件和扩展组件

### 3.2.1 核心组件安装

Name: m2e

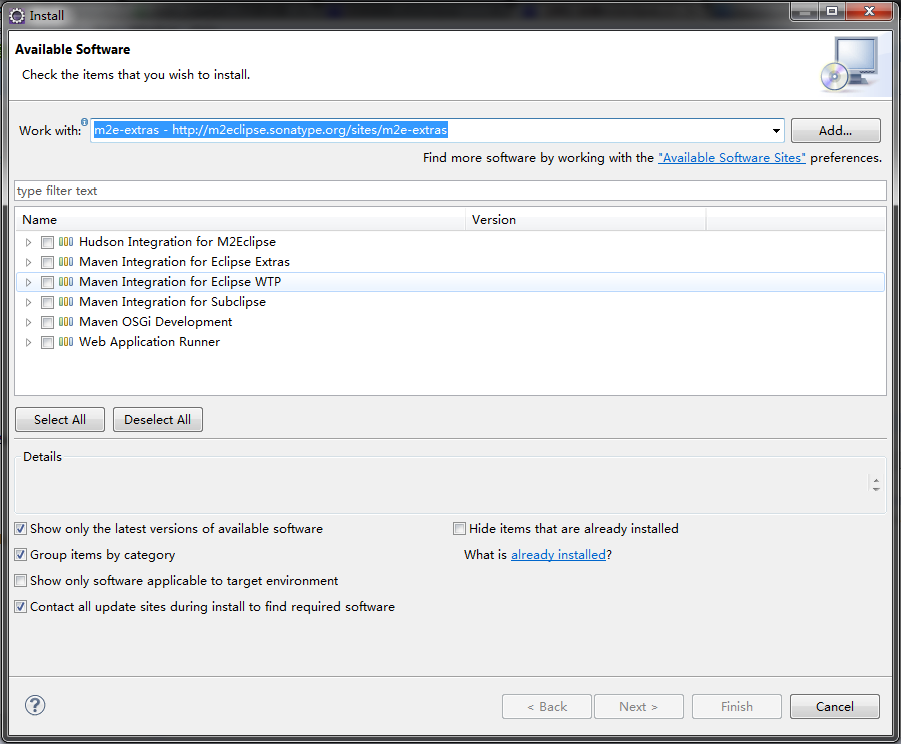
Location: <http://m2eclipse.sonatype.org/sites/m2e>



### 3.2.2扩展组件安装

Name： m2e-extras

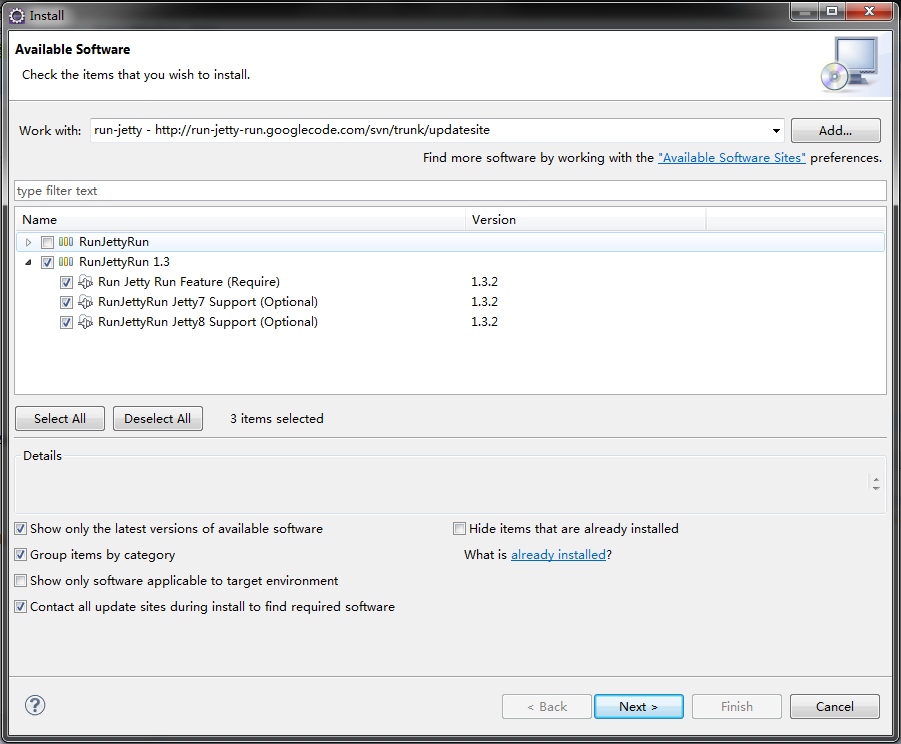
Location：<http://m2eclipse.sonatype.org/sites/m2e-extras>



## Jetty插件安装

Name：run-jetty-run

Location：<http://run-jetty-run.googlecode.com/svn/trunk/updatesite>



## 替换Eclipse内置的Maven

# 目录结构

maven提倡使用共同的标准目录结构，使开发人员能在熟悉了一个Maven工程后，对其他的Maven工程也能清晰了解。

Maven推荐的目录结构如下：

src/main/java--Application/Library sources（源码目录）

src/main/resources--Application/Library resources（资源目录）

src/main/filters--Resource filter files

src/main/assembly--Assembly descriptors

src/main/config--Configuration files

src/main/scripts--Application/Library scripts（js脚本文件）

src/main/webapps--Web application sources（web目录，相当于webcontent）

src/test/java--Test sources（测试源码目录）

src/test/resources--Test resources（测试资源目录）

src/test/filters--Test resource filter files

src/site—Site

target—编译结果目录

LICENSE.txt--Project's license

NOTICE.txt--Notices and attributions required by libraries that the project depends on

README.txt--Project's readme

在顶级目录上是工程的描述文件pom.xml，另外还包括提供给最终用户的文件，如，README.txt, LICENSE.txt等等。

顶级目录还包括两个子目录：src，target。顶级目录下可能出现的其他目录仅仅是CVS或.svn和其他多模块工程的工程目录，最好不要再有其他目录。

Target目录是所有工程编译构建的输出目录。

Src目录包含所有工程的源码文件，配置文件，资源文件等等。它下面的子目录一般包含main（主要的工程源文件），test（测试文件），site（项目站点文件）。

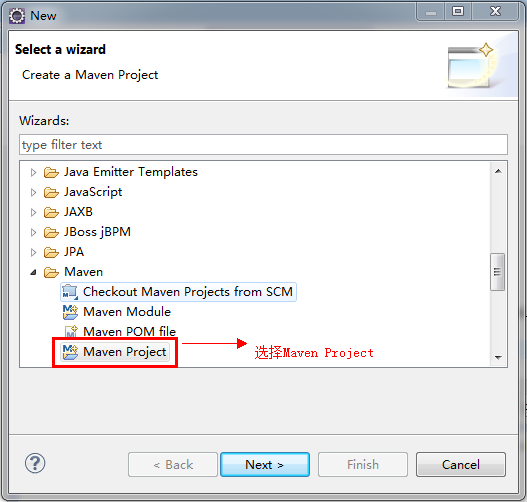
摘自：

http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

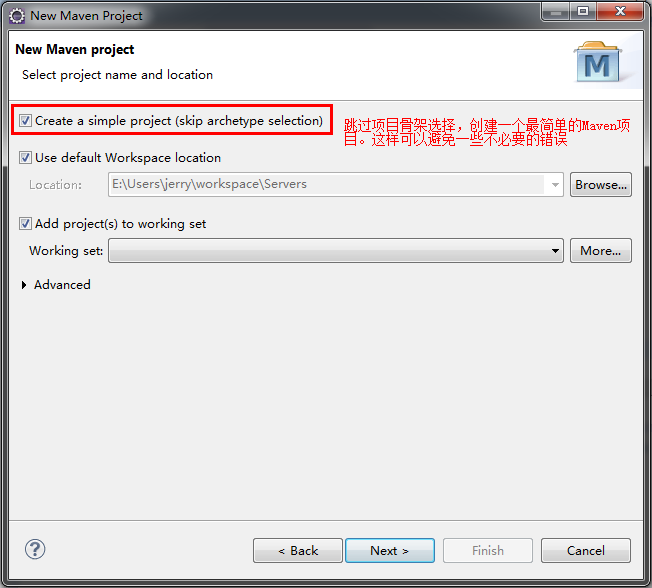
# 在Eclipse中创建Maven项目

* Maven项目创建

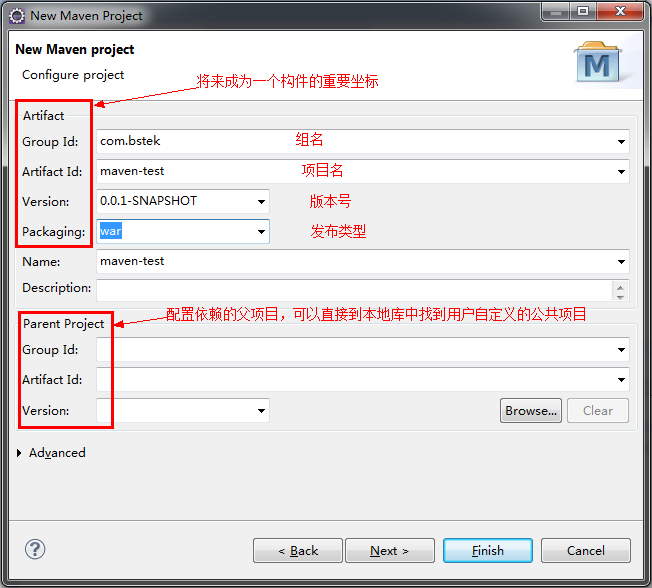
File->new->other->Maven->Maven Object

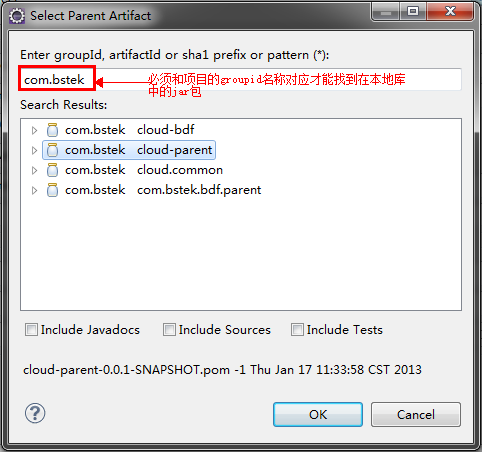
****

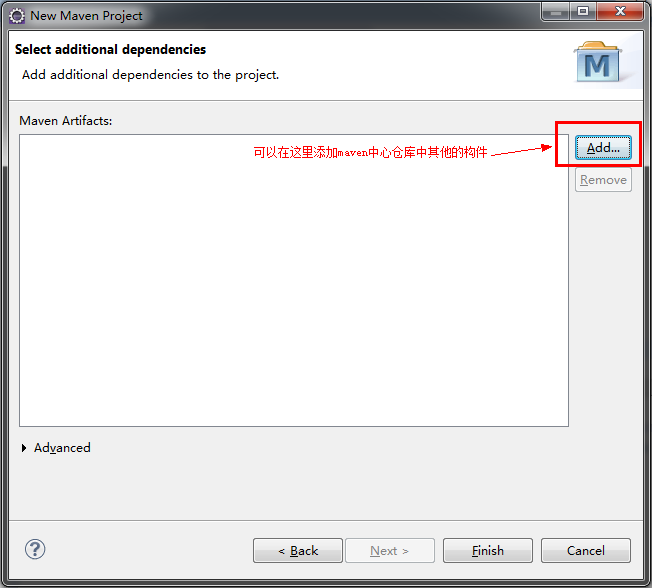
勾上Create a simple project这样可以跳过项目骨架选择

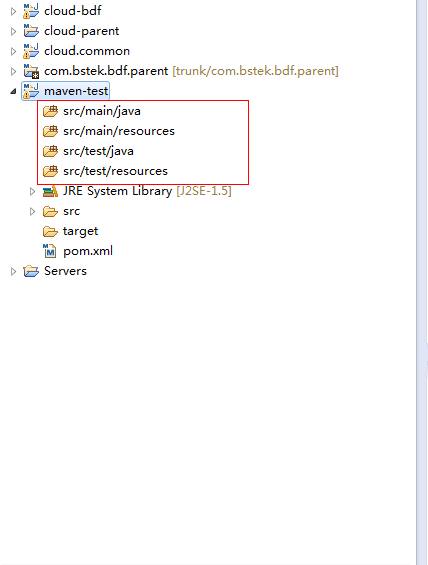
****

填写GroupId和Artifact Id

****

****

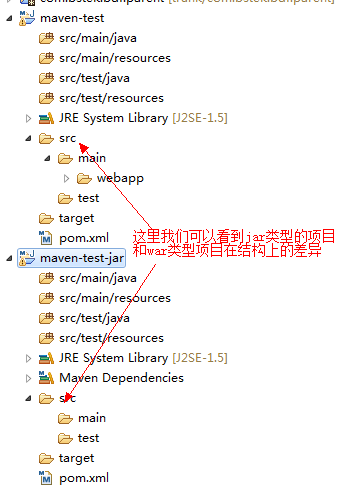
****

****

* 运行调试

通过上一步中，我们创建一个maven-test项目，该项目打包类型为war。也就是说我们以这个项目作为发布的项目，下面我新建一个打包类型为jar类型的maven-test-jar，编写源代码然后在maven-test项目上运行调试。

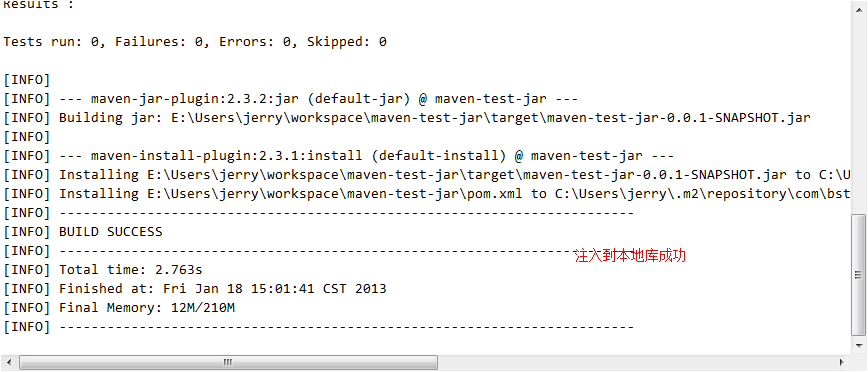
1. 新建maven项目，选择打包类型为jar，同时配置之前配置好的pom类型的parent项目，以便获取到项目开发中常用的构件。如图下图所示：



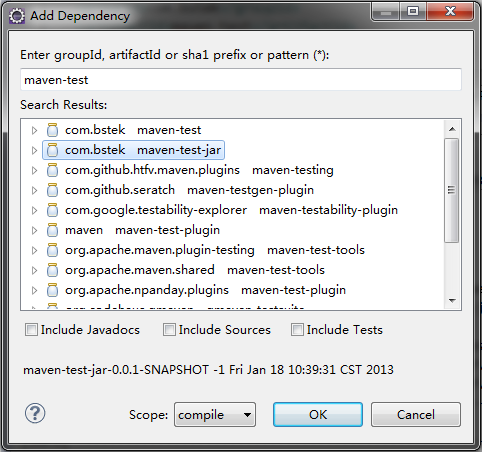
1. 新建的maven-test-jar项目由于添加了继承关系，所以要更新一下该项目的依赖关系。

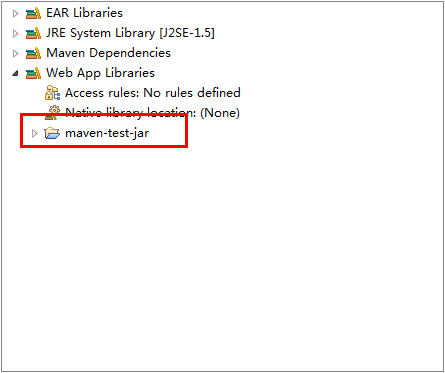
在项目名称右击->maven->Update Dependencies

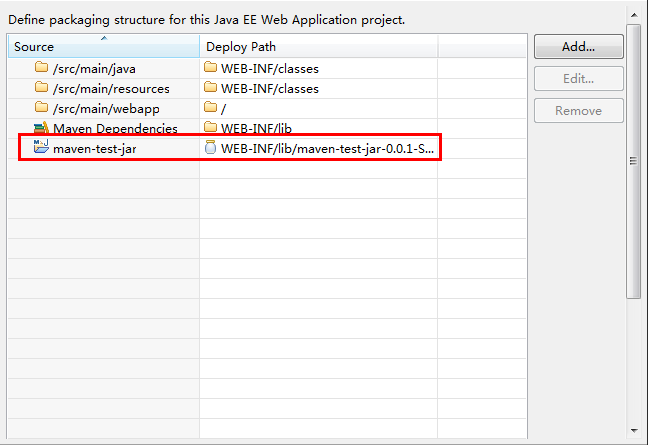
1. 将maven-test-jar注入到本地库中



1. 在maven-test项目中添加依赖关系并更新依赖关系，然后我们通过项目的属性查看的Deployment Assembly中可以看到项目关系。



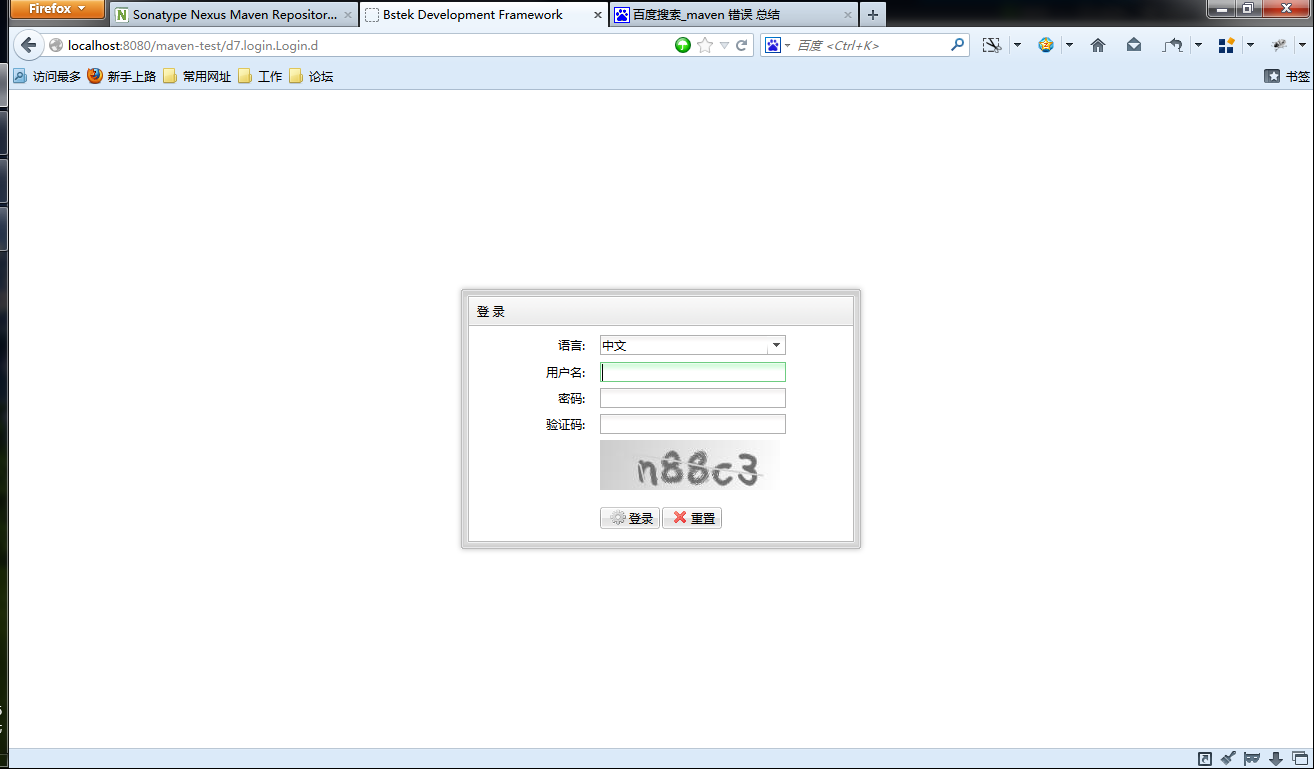




1. 更行maven-test项目的依赖之后，我们maven-test-jar项目添加一些代码，启动maven-test项目，在浏览器中输入地址：

<http://localhost:8080/maven-test/>

即可访问到jar类型项目中的源代码，同时也可以进行代码调试。



# 项目构建生命周期

Maven是围绕着构建生命周期概念设计的。也就是说构建或发布的过程已经被清晰的定义了。

当我们使用Maven构建工程时，我们只需要了解几个Maven定义好的命令即可，其他的工作则交给POM来完成。

Maven提供的构建生命周期列表：

validate

validate the project is correct and all necessary information is available.

generate-sources

generate any source code for inclusion in compilation.

process-sources

process the source code, for example to filter any values.

generate-resources

generate resources for inclusion in the package.

process-resources

copy and process the resources into the destination directory, ready for packaging.

compile

compile the source code of the project.

process-classes

post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.

generate-test-sources

generate any test source code for inclusion in compilation.

process-test-sources

process the test source code, for example to filter any values.

generate-test-resources

create resources for testing.

process-test-resources

copy and process the resources into the test destination directory.

test-compile

compile the test source code into the test destination directory

test

run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.

package

take the compiled code and package it in its distributable format, such as a JAR.

pre-integration-test

perform actions required before integration tests are executed. This may involve things such as setting up the required environment.

integration-test

process and deploy the package if necessary into an environment where integration tests can be run.

post-integration-test

perform actions required after integration tests have been executed. This may including cleaning up the environment.

verify

run any checks to verify the package is valid and meets quality criteria.

install

install the package into the local repository, for use as a dependency in other projects locally.

deploy

done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

因此，当我们构建一个项目时，只需要了解自己希望做什么，然后执行以上对应的生命周期即可。

例如，我们希望编译我们的工程。在命令行状态下进入到工程的pom.xml文件所在的目录中，使用命令:mvn compile；希望构建打包我们的工程，使用mvn package即可。

当然了，maven的构建生命周期也是可以扩展和自定义的。

# 插件

## Eclipse常用插件安装

* Mylyn //集成任务管理和上下文管理

Name:Mylyn for Eclipse 3.4 and 3.5

Location:http://download.eclipse.org/tools/mylyn/update/e3.4/

* aspectJ //切面编程插件

Name:aspectJ

Location:http://download.eclipse.org/tools/ajdt/35/update

* Subclipse //SVN版本控制器客户端

Name:Subclipse 1.6.x Update Site

Location:http://subclipse.tigris.org/update\_1.6.x

* WTP //Web 工具平台

Name:The Eclipse Web Tools Platform (WTP) Project update site

Location:http://download.eclipse.org/webtools/updates

* EPP //负责创建Eclipse下载软件包

Name:EPP Packages Repository

Location:http://download.eclipse.org/technology/epp/packages/galileo

# 私服-nexus