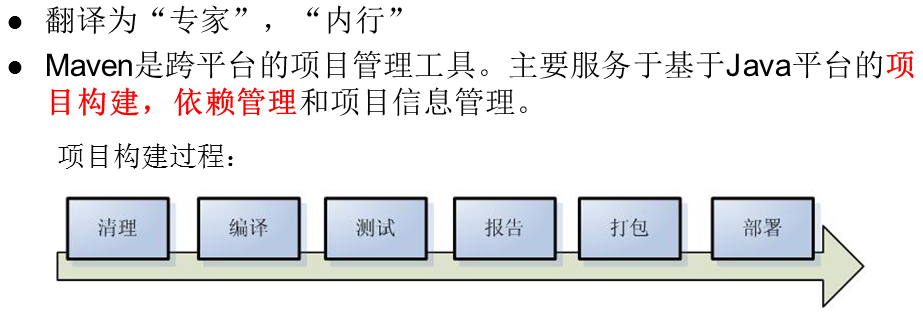
Maven基础

# 1基础概念

## 1.1Maven基础概念

**1)什么是Maven**



* 什么是理想的项目构建？
  + 高度自动化，跨平台，可重用的组件，标准化的
* 什么是依赖？为什么要进行依赖管理？
  + 自动下载，统一依赖管理
* 有哪些项目信息？
  + 项目名称描述等，开发人员信息，开发者信息等

没有学些Maven之前

使用eclipse开发的步骤：

\*1清理(clean)

2 编译(java文件转换为class文件)

3测试

4报告(文档)

5打包(生成jar包或者war包)

6部署(部署到tomcat)

\*复制(SpringMVC+MyBatis的jar包)

学习完Maven

使用eclipse开发的步骤：

下面6个步骤可以使用Maven的一步构建（只需要一个命令即可）

\*1清理(clean)

2 编译(java文件转换为class文件)

3测试

4报告(文档)

5打包(生成jar包或者war包)

6部署(部署到tomcat)

\*通过一个配置(pom.xml)自动从互联网导入jar包

## 1.2Maven的优势

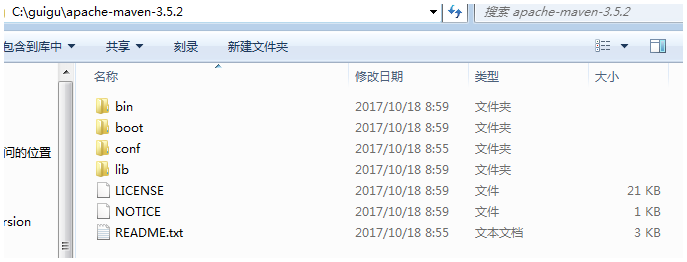
* 为什么使用Maven？
* IDE？Eclipse？
  + 手工操作较多，编译、测试、部署等工作都是独立的，很难一步完成
  + 每个人的IDE配置都不同，很容易出现本地代码换个地方编译就出错
* Ant？
  + 没有一个约定的目录结构
  + 必须明确让ant做什么，什么时候做，然后编译，打包
  + 没有生命周期，必须定义目标及其实现的任务序列
  + 没有集成依赖管理
* Maven？
  + 拥有约定，知道你的代码在哪里，放到哪里去
  + 拥有一个生命周期，例如执行 mvn install 就可以自动执行编译，测试，打包等构建过程
  + 只需要定义一个pom.xml,然后把源码放到默认的目录，Maven帮你处理其他事情
  + 拥有依赖管理，仓库管理

## 1.3Maven的配置的安装

* 安装Maven
  + 确认jdk是否已安装？
  + 下载Maven
    - 地址：<http://maven.apache.org/download.html>
    - 版本：Maven3.0
  + 安装Maven

1. 下载Maven

2）Maven的目录结构



1. 配置环境变量

MAVEN\_HOME =C:\guigu\apache-maven-3.5.2

Path= C:\guigu\apache-maven-3.5.2\bin

4)校验是否安装成功

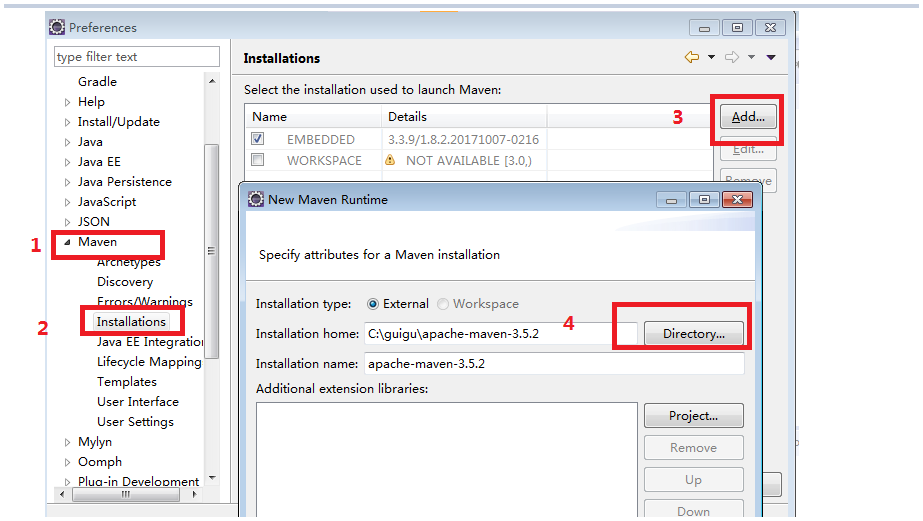
此时验证需要配置JAVA\_HOME



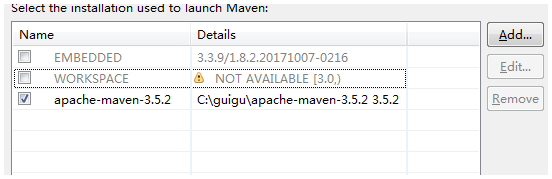
## 1.4Maven在Eclipse中的配置

**1）指定eclipse使用自定义的maven版本信息**

Windows--->preference--->maven

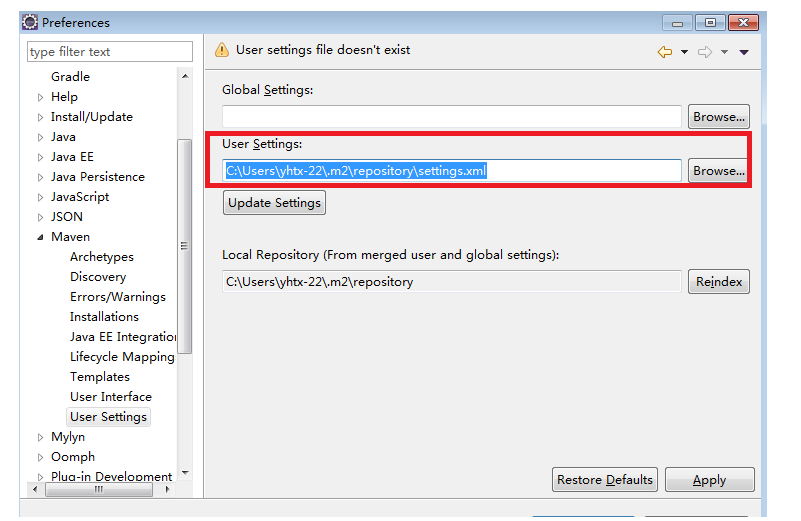


选中当前的版本信息



**2）指定用户的配置setting.xml**

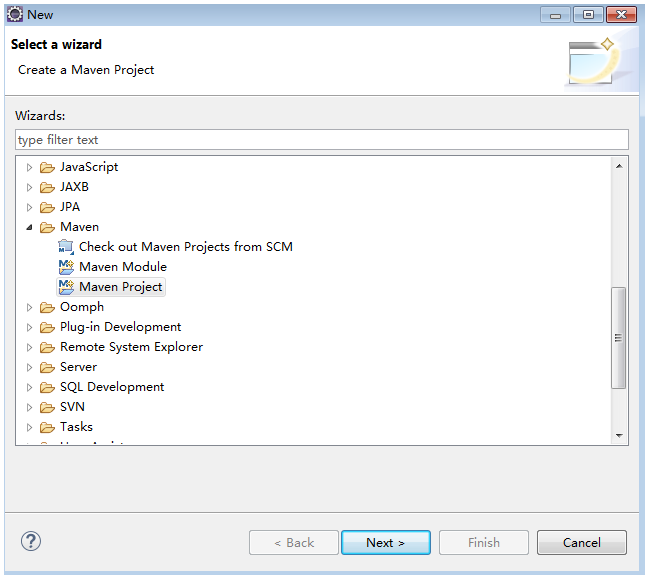
把C:\guigu\apache-maven-3.5.2\conf\setting.xml文件放置到C:\Users\yhtx-22\.m2



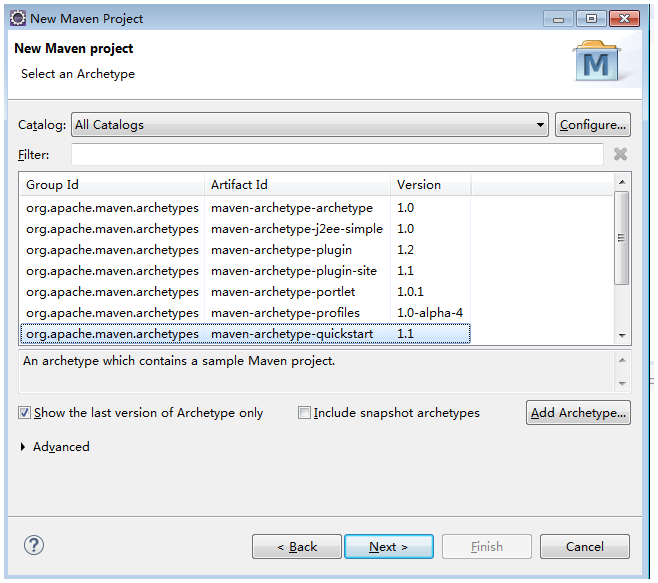
# 2 Maven入门程序

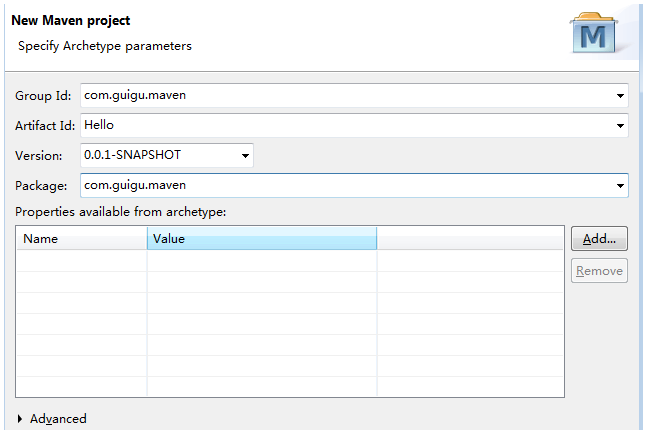
## 2.1第一个Maven项目Hello

### 2.1.1新建项目

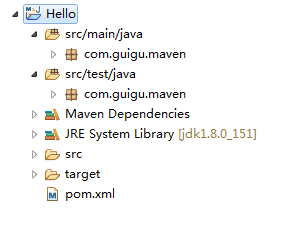


选择Maven骨架





建立完成的目录结构



遵从Maven约定

src/main/java —— 存放项目的.java文件

src/main/resources —— 存放项目资源文件，如spring, hibernate配置文件

src/test/java —— 存放所有测试.java文件，如JUnit测试类

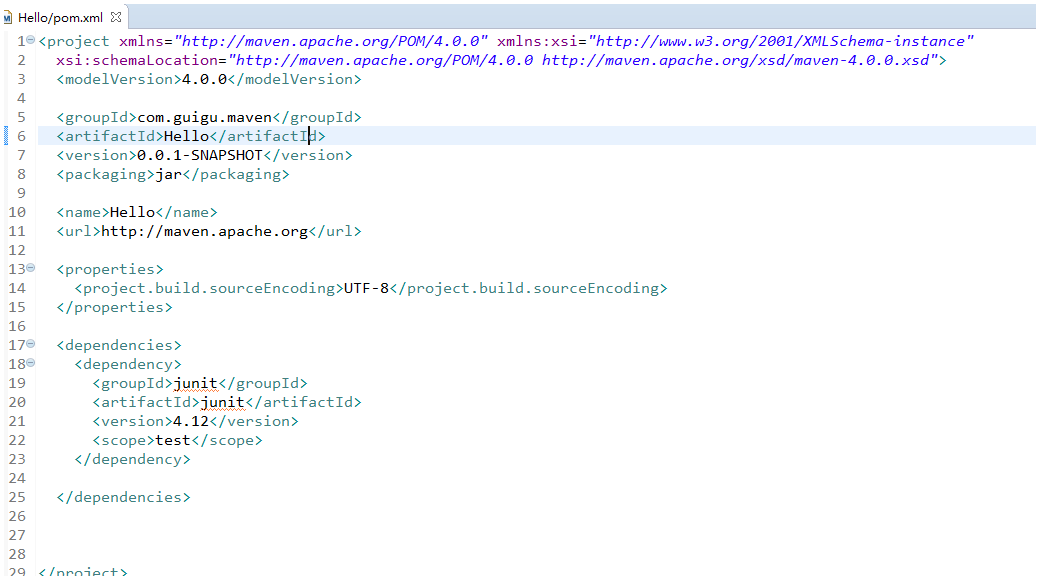
src/test/resources —— 测试资源文件

target —— 项目输出位置

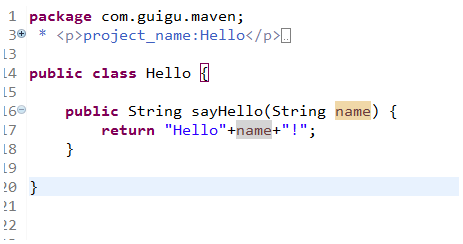
pom.xml

### 2.1.2 pom.xml

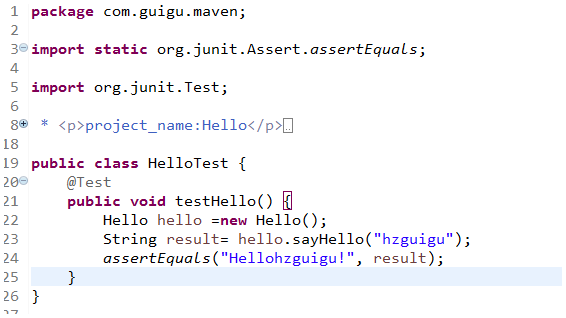
可以更换Junit的jar包查看变化



### 2.1.3Hello.java



### 2.1.4HelloTest.java



### 2.1.5执行程序

选中pom.xml 右键 --->run as -->maven bulide

依次执行以下命令 查看根据目录的变化

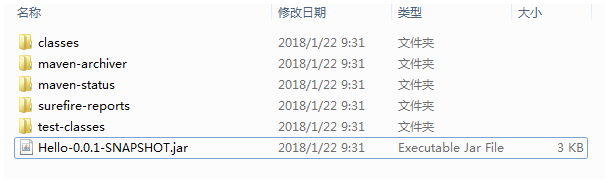
A)执行compile命令 查看根目录变化 编译程序为.class文件

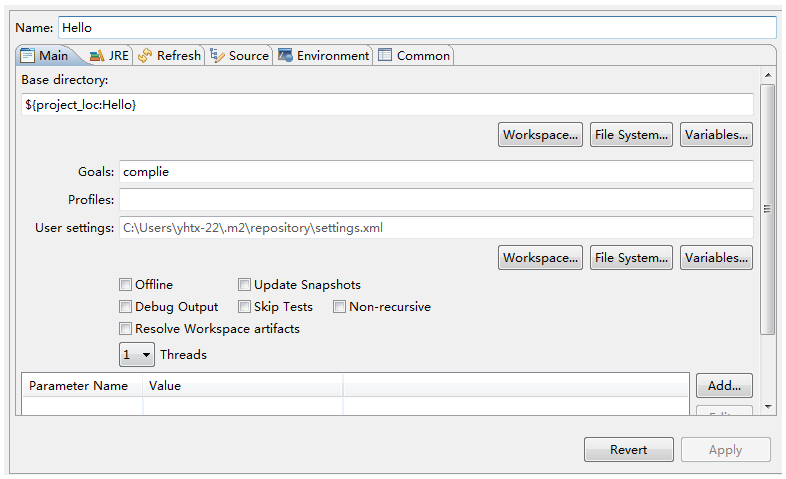
B)执行clean 命令 清理target所有的内容

C) 执行 clean compile

D) 执行 clean test 执行测试程序

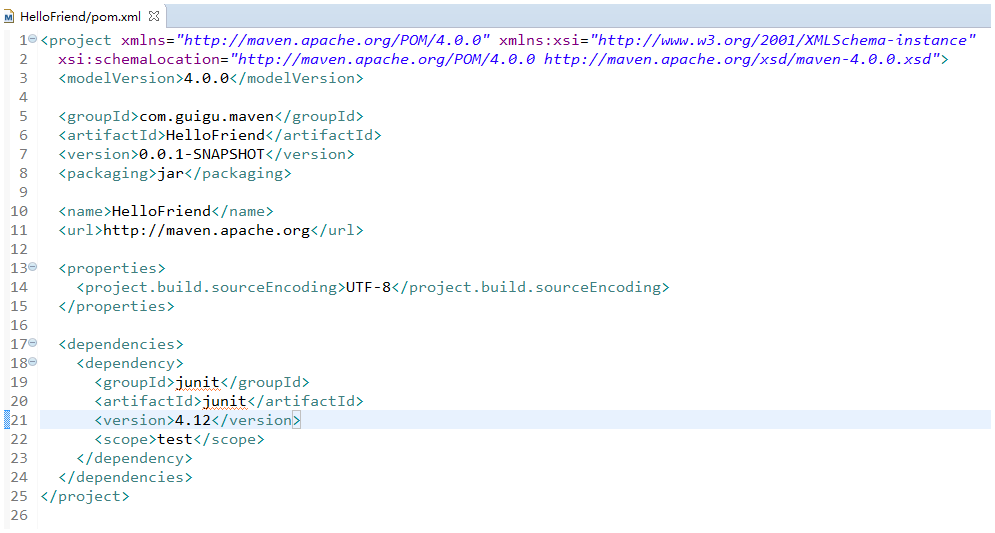
E) 执行clean package 打包数据





## 2.2第二个Maven项目HelloFriend

### 2.2.1Pom.xml



### 2.2.2HelloFriend.java

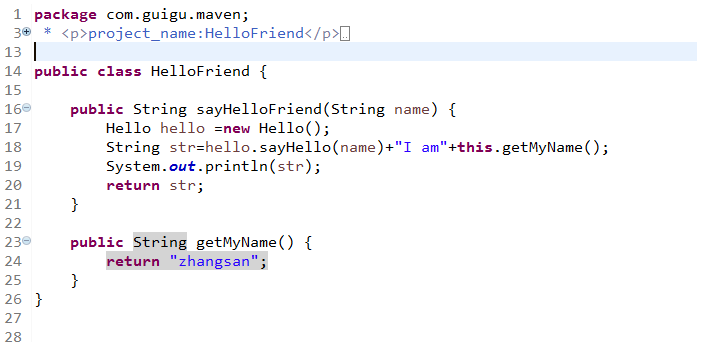
第一步把Hello工程工程部署

执行install命令即可部署

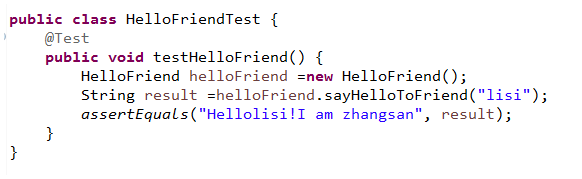
第二步在HelloFriend中引用Hello工程



在工程中即可正常引用内容



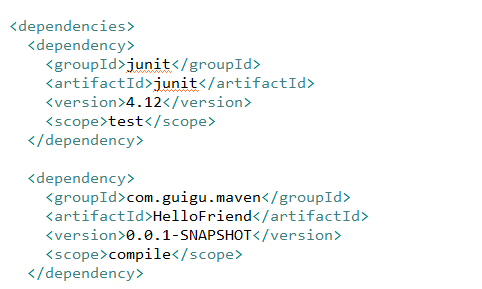
### 2.2.3HelloFriendTest.java



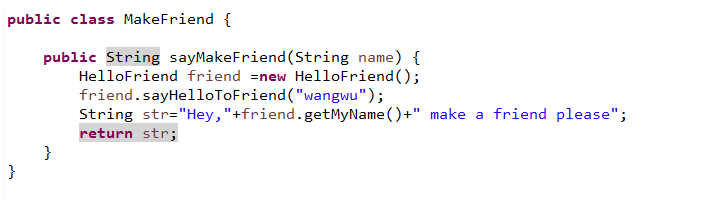
同样通过install安装到本地仓库下

## 2.3第三个Maven项目MakeFriend

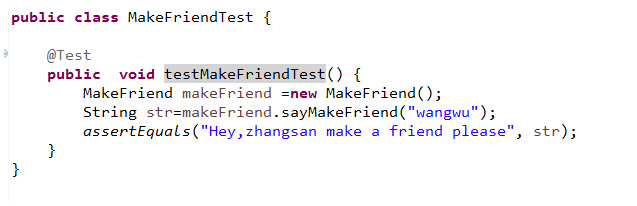
### 2.3.1Pom.xml



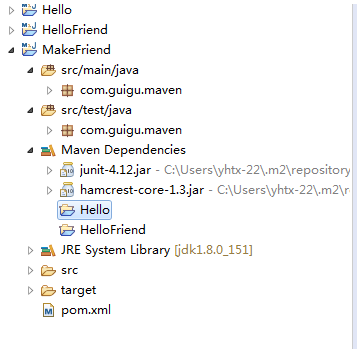
### 2.3.2MakeFriend.java



### 2.3.3MakeFriendTest.java

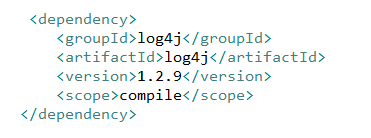


查看Maven的依赖

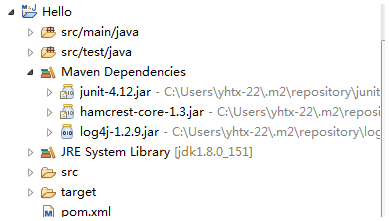


## 2.4间接依赖和直接依赖

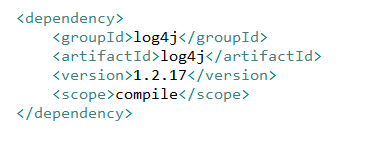
1）在Hello中引入log4j.jar



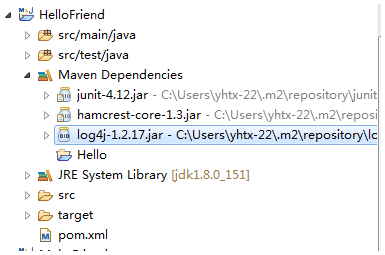
查看Maven依赖



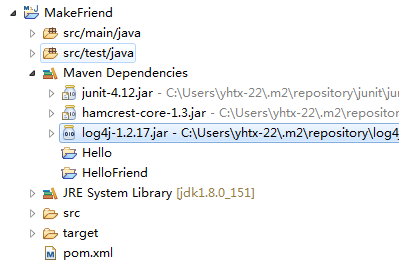
2）在HelloFriend中 添加

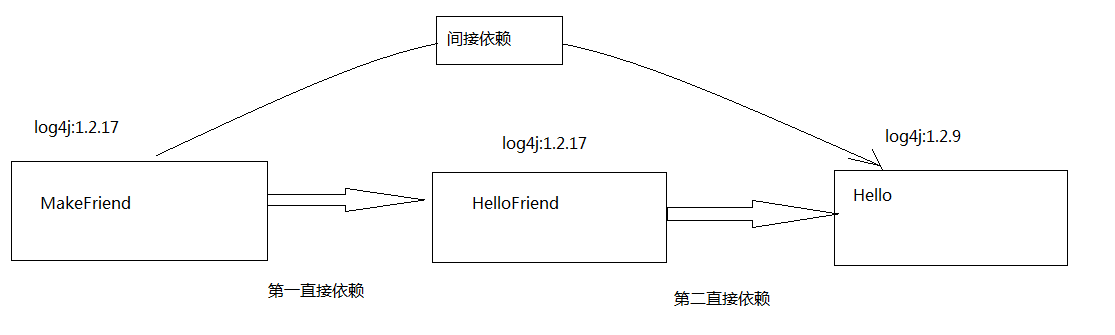


并查看Maven的依赖



3）查看MakeFriend的变化



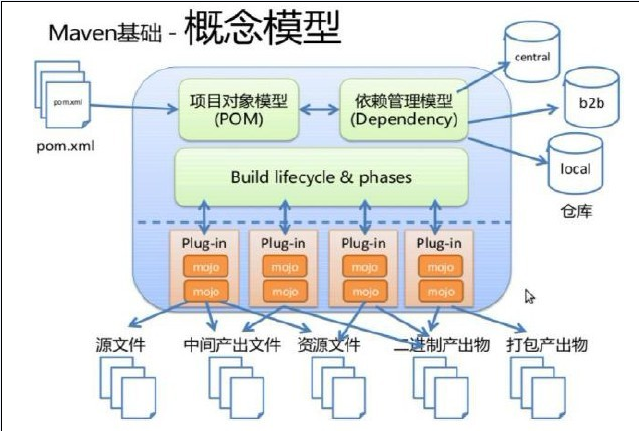


原则是

1. 路径最近者优先

# 3Maven的核心概念

## 3.1Maven的概念模型



（1）Maven的核心仅仅定义了抽象的生命周期，具体的任务都是交由**插件**完成的

（2）每个插件都能实现多个功能，每个功能就是一个插件目标

（3）Maven的生命周期与插件目标相互绑定，以完成某个具体的构建任务

插件位置： C:\Users\yhtx-22\.m2\repository\org\apache\maven\plugins

<build>

<plugins>

<plugin>

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

<artifactId>maven-source-plugin</artifactId>

<version>2.2.1</version>

<executions>

<execution>

<goals>

<goal>jar-no-fork</goal>

</goals>

<!--

verify的生命周期在package之后，install之前，

即如果执行package则不会执行verify，但是如果执行install则一定要执行verify

-->

<phase>

**verify**

</phase>

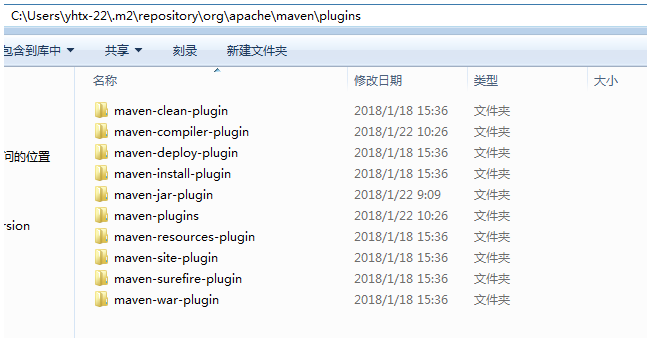
</execution>

</executions>

</plugin>

</plugins>

</build>



如果执行clean package 可以看到target包中生成的jar包 里面放置的是.class文件

如果执行clean install 可以看到生成的源码文件

## 3.2Maven的生命周期

* 何为生命周期？
  + Maven生命周期就是为了对所有的构建过程进行抽象和统一
  + 包括项目清理，初始化，编译，打包，测试，部署等几乎所有构建步骤
* Maven三大生命周期
  + clean：清理项目的
  + default：构建项目的
  + site：生成项目站点的

生命周期Maven有三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，这三套生命周期分别是：

**Clean Lifecycle 在进行真正的构建之前进行一些清理工作。**

**Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。**

**Site Lifecycle 生成项目报告，站点，发布站点。**

再次强调一下它们是相互独立的，你可以仅仅调用clean来清理工作目录，仅仅调用site来生成站点。当然你也可以直接运行 mvn clean install site 运行所有这三套生命周期。

**Clean Lifecycle:**

clean生命周期每套生命周期都由一组阶段(Phase)组成，我们平时在命令行输入的命令总会对应于一个特定的阶段。比如，运行mvn clean ，这个的clean是Clean生命周期的一个阶段。有Clean生命周期，也有clean阶段。Clean生命周期一共包含了三个阶段：

pre-clean 执行一些需要在clean之前完成的工作

**clean 移除所有上一次构建生成的文件**

post-clean 执行一些需要在clean之后立刻完成的工作

mvn clean 中的clean就是上面的clean，在一个生命周期中，运行某个阶段的时候，它之前的所有阶段都会被运行，也就是说，mvn clean 等同于 mvn pre-clean clean ，如果我们运行 mvn post-clean ，那么 pre-clean，clean 都会被运行。这是Maven很重要的一个规则，可以大大简化命令行的输入。

**Default Lifecycle:**

Default生命周期Default生命周期是Maven生命周期中最重要的一个，绝大部分工作都发生在这个生命周期中。这里，只解释一些比较重要和常用的阶段：

validate

generate-sources

process-sources

generate-resources

process-resources 复制并处理资源文件，至目标目录，准备打包。

**compile 编译项目的源代码。**

process-classes

generate-test-sources

process-test-sources

generate-test-resources

process-test-resources 复制并处理资源文件，至目标测试目录。

test-compile 编译测试源代码。

process-test-classes

**test 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。**

prepare-package

**package 接受编译好的代码，打包成可发布的格式，如 JAR 。**

pre-integration-test

integration-test

post-integration-test

verify

**install 将包安装至本地仓库，以让其它项目依赖。**

**deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。**

**运行任何一个阶段的时候，它前面的所有阶段都会被运行，这也就是为什么我们运行mvn install 的时候，代码会被编译，测试，打包。**此外，Maven的插件机制是完全依赖Maven的生命周期的，因此理解生命周期至关重要。

Site Lifecycle:

Site生命周期

pre-site 执行一些需要在生成站点文档之前完成的工作

site 生成项目的站点文档

post-site 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备

site-deploy 将生成的站点文档部署到特定的服务器上

这里经常用到的是site阶段和site-deploy阶段，用以生成和发布Maven站点，这可是Maven相当强大的功能，Manager比较喜欢，文档及统计数据自动生成，很好看。

通过site可以生成站点信息

<!-- 项目信息 -->

<description>杭州归谷培训中心</description>

<!-- 项目创建年月 -->

<ciManagement>

</ciManagement>

<inceptionYear>2018-1-22</inceptionYear>

<!-- 公司信息 -->

<organization>

<name>杭州归谷培训中心</name>

<url>http://www.hzguigu.com/</url>

</organization>

<!-- 研发人员列表 -->

<developers>

<developer>

<id>liuhuan</id>

<name>刘欢</name>

<email>liuhuan@163.com</email>

<roles>

<role>研发经理</role>

<role>研发工程师</role>

</roles>

</developer>

</developers>

<!-- 贡献人列表 -->

<contributors>

<contributor>

<name>张无忌</name>

<email>zhangwuji@163.com</email>

<roles><role>研发工程师</role></roles>

</contributor>

<contributor>

<name>郭靖</name>

<email>guojing@163.com</email>

<roles><role>研发工程师</role></roles>

</contributor>

</contributors>

<!-- 许可声明 -->

<licenses>

<license>

<name>许可</name>

<url>http://mis.guigu.cn</url>

<comments>\*\*\*\*</comments>

</license>

</licenses>

<!-- 版本控制连接 -->

<scm>

<connection>scm:svn:svn://localhost/test/tags/mis-parent-1.0/test</connection>

<developerConnection>scm:svn:svn://localhost/test/tags/mis-parent-1.0/trunk</developerConnection>

</scm>

<!--问题跟踪系统 -->

<issueManagement>

<system>Streber</system>

<url>http://localhost/index.php</url>

</issueManagement>

可以配置相关的站点信息

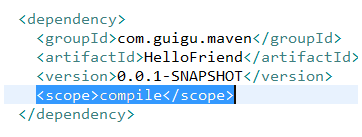
## 3.3核心概念

### 3.3.1 坐标

* 什么是坐标？
  + 在平面几何中坐标（x,y）可以标识平面中唯一的一点
* Maven坐标主要组成
  + groupId：定义当前Maven项目隶属项目
  + artifactId：定义实际项目中的一个模块（主类）
  + version：定义当前项目的当前版本
  + packaging：定义该项目的打包方式（jar包或者war包）
* Maven为什么使用坐标？
  + Maven世界拥有大量构建，我们需要找一个用来唯一标识一个构建的统一规范
  + 拥有了统一规范，就可以把查找工作交给机器

### 3.3.2依赖管理

#### 3.3.2.1依赖配置



其中<scope>compile</scope> 表示范围

#### 3.3.2.2依赖的范围



#### 3.3.2.3传递性依赖和可选依赖以及排除依赖

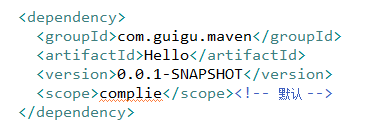
列是第一个直接依赖 行是第二直接依赖

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | compile | test | provided | runtime |
| compile | compile | - | - | runtime |
| test | test | - | - | test |
| provided | provided | - | provided | provided |
| runtime | runtime | - | - | runtime |

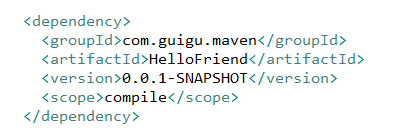
**1）传递性依赖：路径最近者优先**

**案例一：**

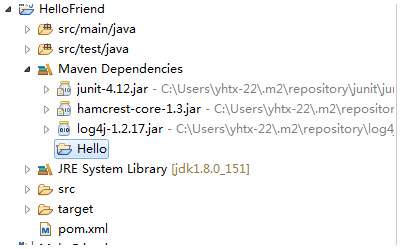
在HelloFriend的pom.xml中配置

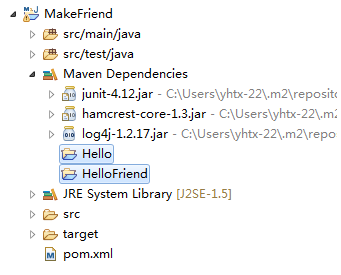


在MakeFriend的pom.xml



可以查看Maven的依赖结果



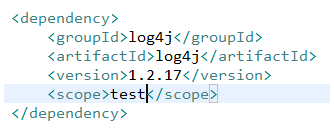


HelloFriend 和MakeFriend都属于第一直接依赖范围都是compile 。HelloFriend和Hello是第二直接依赖 范围是compile 。

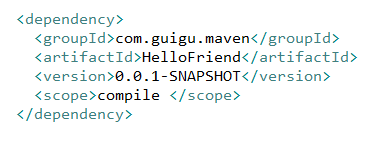
通过查阅表 MakeFriend和Hello是间接依赖。 范围compile，所以会在MakeFriend中显示出来。

**案例二：**

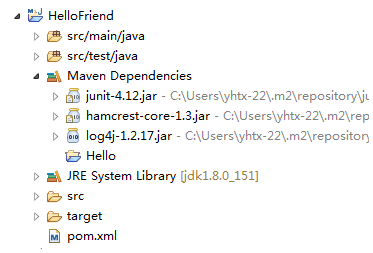
如果在HelloFriend的pom.xml

 HelloFrei

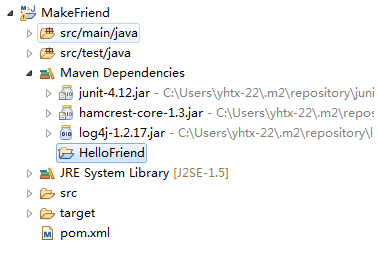
在MakeFriend的pom.xml



查看Maven依赖的结果是



通过查看MakeFriend中没有Hello的间接依赖

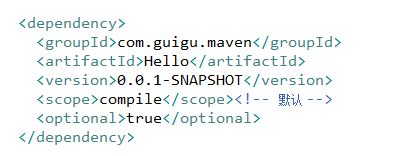


1. **可选依赖不被传递**

方式一）

在MakeFriend中不需要引用Hello这个项目 可以在HelloFriend中通过可选依赖配置不再传递给MakeFriend

在HelloFriend中的pom.xml中通过optional 标识当前引用不再继续往下传递



1. **排除依赖**

如果在MakeFriend中只想引入HelloFriend 不需要Hello包 可以通过排除依赖配置



### 3.3.3常见的仓库管理

**推荐几个好的 Maven 常用仓库网址：**[**http://mvnrepository.com/**](http://mvnrepository.com/)[**http://search.maven.org/**](http://search.maven.org/)  
[**http://repository.sonatype.org/content/groups/public/**](http://repository.sonatype.org/content/groups/public/)  
[**http://people.apache.org/repo/m2-snapshot-repository/**](http://people.apache.org/repo/m2-snapshot-repository/)  
[**http://people.apache.org/repo/m2-incubating-repository/**](http://people.apache.org/repo/m2-incubating-repository/)

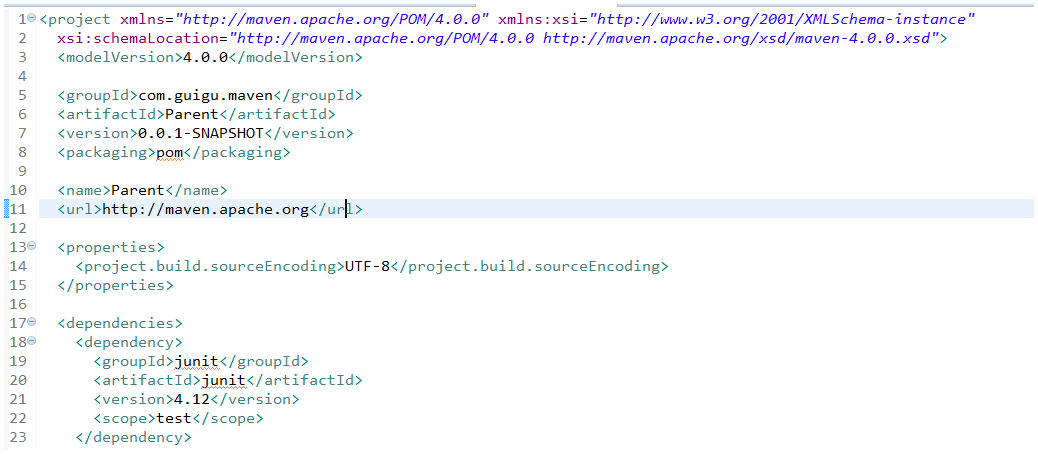
# 4Maven的聚合和继承

## 4.1继承

* 何为继承？
  + 继承为了消除重复，我们把很多相同的配置提取出来
  + 例如：grouptId，version等

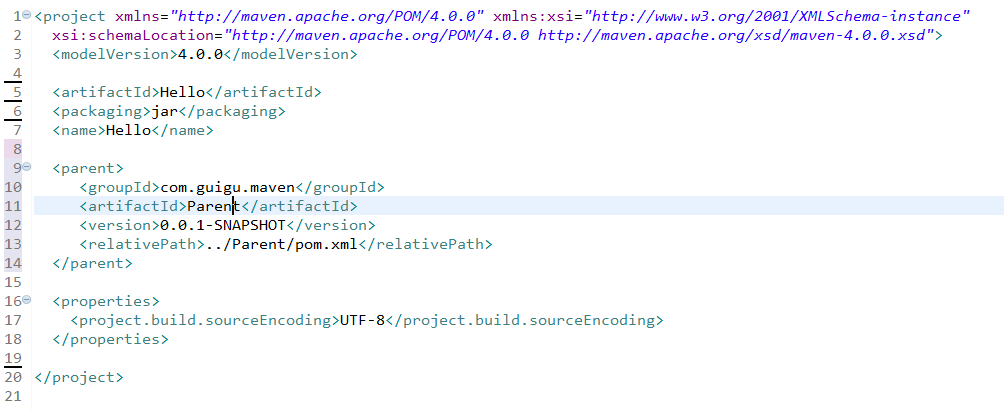
第一步新建工程Parent

第二步 在parent的pom.xml提取公共数据

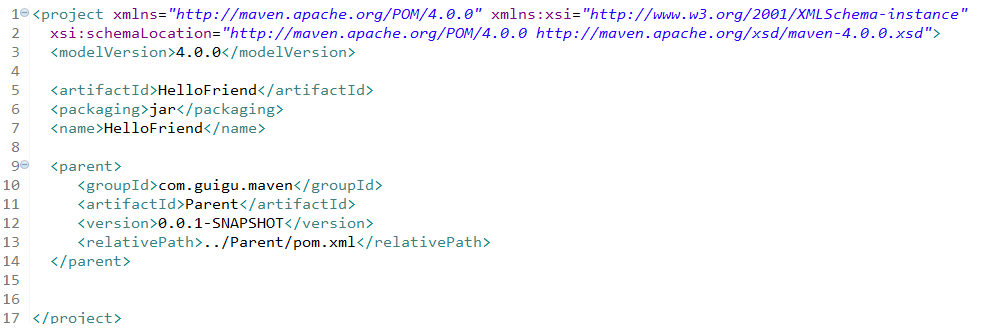


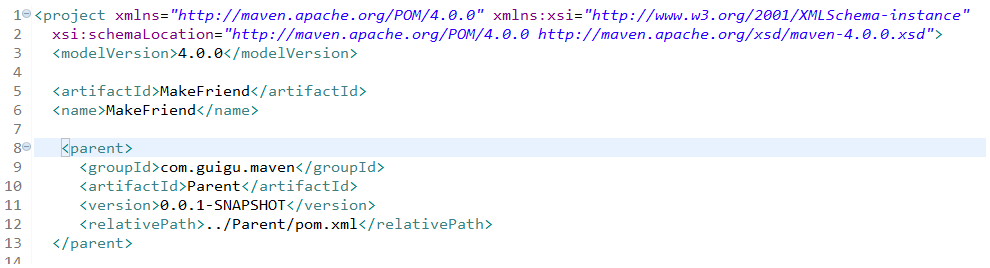


第三步在Hello的pom.xml中配置

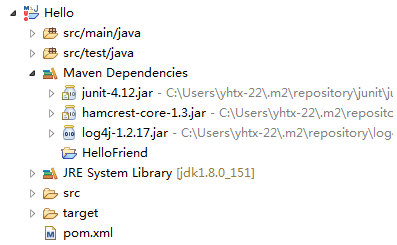


第四步:把HelloFriend和MakeFriend修改如下





问题：此时Hello项目中有HelloFriend 在Hello中是不需要HelloFriend的数据



解决办法 在parent的pom.xml添加依赖管理

<dependencyManagement>

<dependencies>

</dependencies>

</dependencyManagement>

执行clean

查阅Maven依赖会发现所有的依赖全部都不存在 。

解决方案：

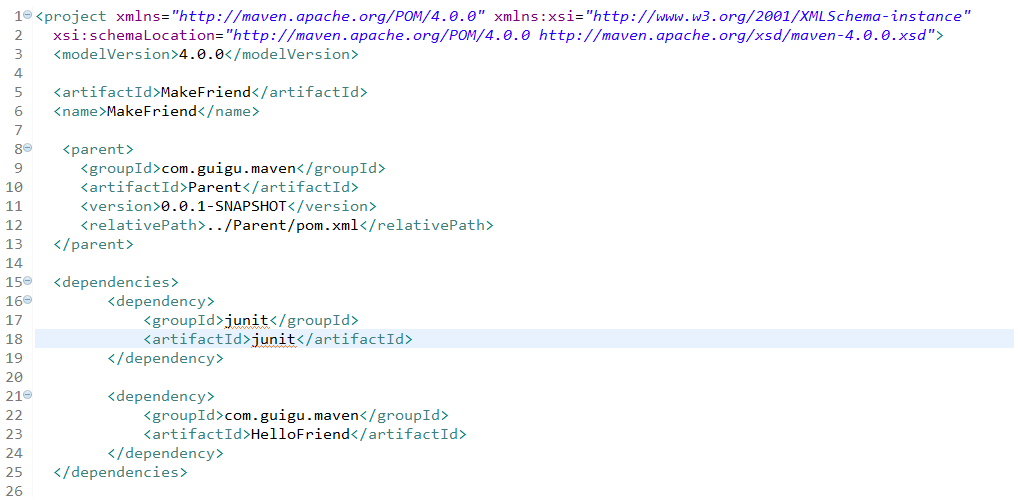
1）在Hello项目的pom.xml中添加依赖



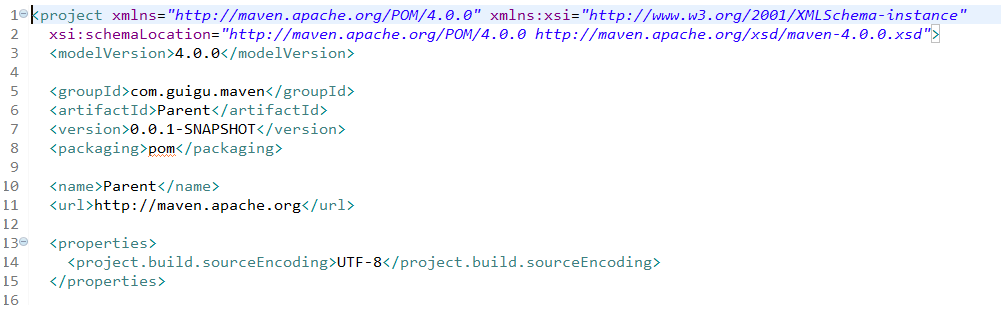
2）HelloFriend



3)MakeFriend



4)Parent

## 4.2聚合

如果想要执行每个子工程都需要 编辑---测试—打包—部署

可以在父工程中统一定义

