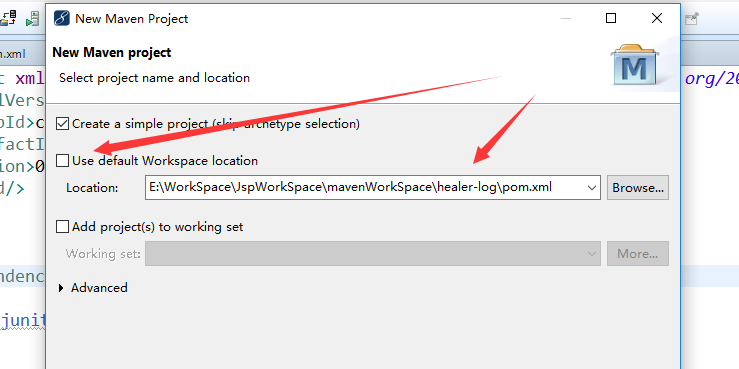
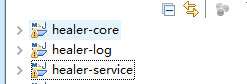
# 0、不能够新建项目、如果不勾选下面这个会报错，所以即使和下面的没有关系，也要选择上,有时候项目出错了，只要用eclipse更新下项目就好了



# 这里的三个项目还没有走向聚合

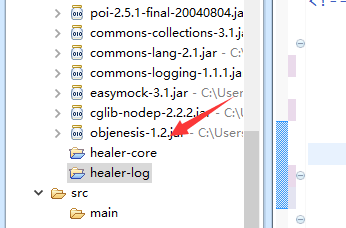


# 1、healer-service依赖前两者

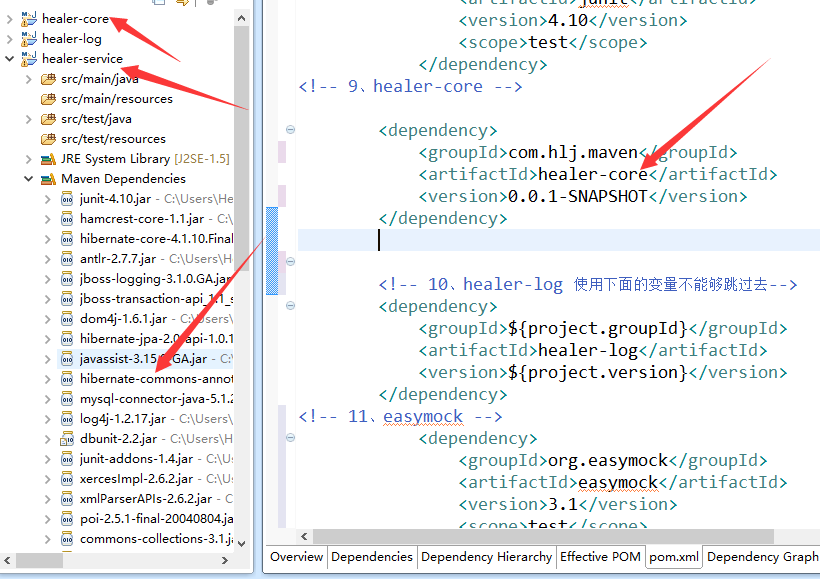
## 1、如果使用${project.groupId} 使用ctrl+鼠标则不能够移动到所代表的项目的pom中，而直接使用名字可以移动到项目中

|  |
| --- |
| <!-- 9、healer-core -->    <dependency>  <groupId>com.hlj.maven</groupId>  <artifactId>healer-core</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>    <!-- 10、healer-log 使用下面的变量不能够跳过去-->  <dependency>  <groupId>${project.groupId}</groupId>  <artifactId>healer-log</artifactId>  <version>${project.version}</version>  </dependency> |

## 2、结果



## 3、如果引入了上面的依赖，则对于这二者pom所引入的包也会自动帮我们引入进来，

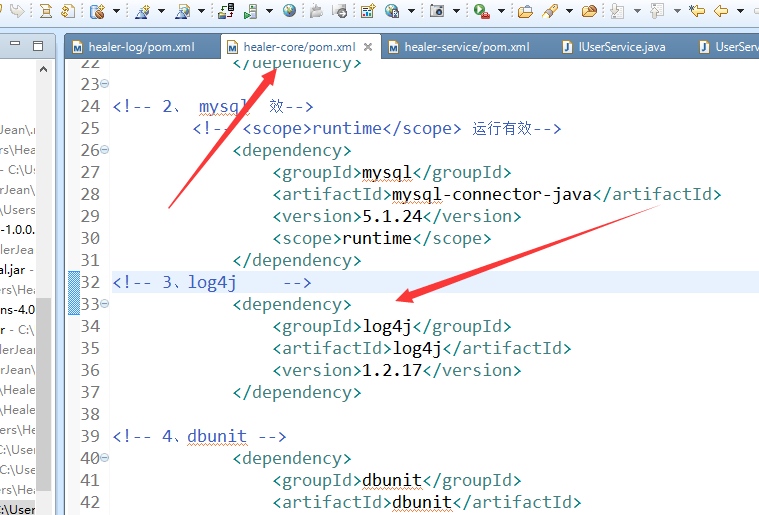


### 1、但是如果依赖的版本有了scope依赖则scope不会传递下去，这个和父类依赖管理器是不一样的哦，父类依赖管理器是可以传递的

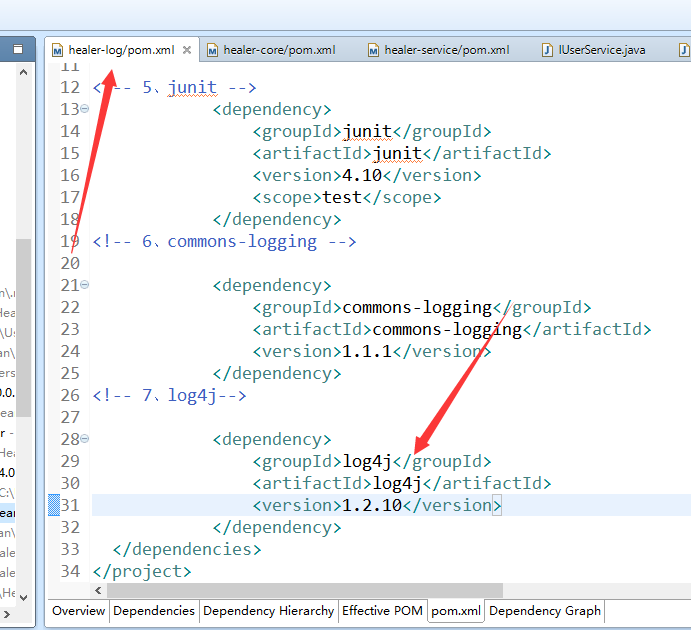
### 2、依赖级别问题（与version无关）

#### 1、healer-core和healer-log 级别（依赖的继承级别）一样，所以对于从这二者来的log4j会选择先写入pom文件中的healer-core（本例）中的log4j的版本

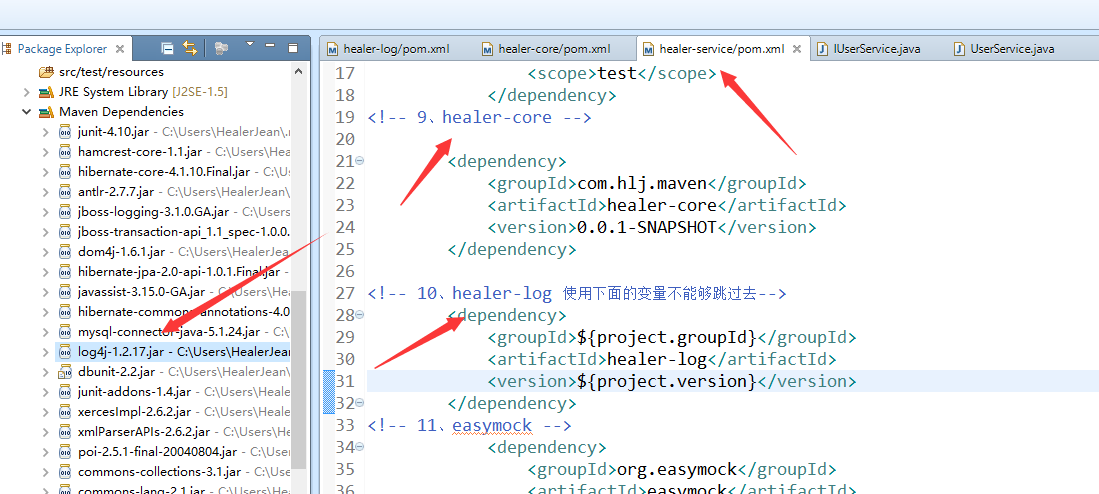
##### 1、healer-core log4j 1.2.17



##### 2、healer-log log4j 1.2.10



##### 3、healer-service log4j 1.2.17 (core先写，所以用的是core中的依赖)



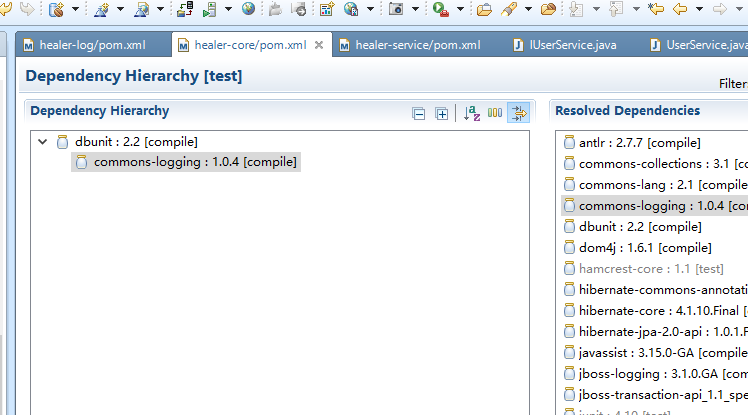
#### 2、此时如果在service中的pom中重新添加标准的commons-logging的依赖，则会舍弃上面从healer-log中的依赖版本。使用现在这个级别的依赖，因为这个级别是0（距离healer-service是最近的）。

|  |
| --- |
| <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.1.1</version>  </dependency> |

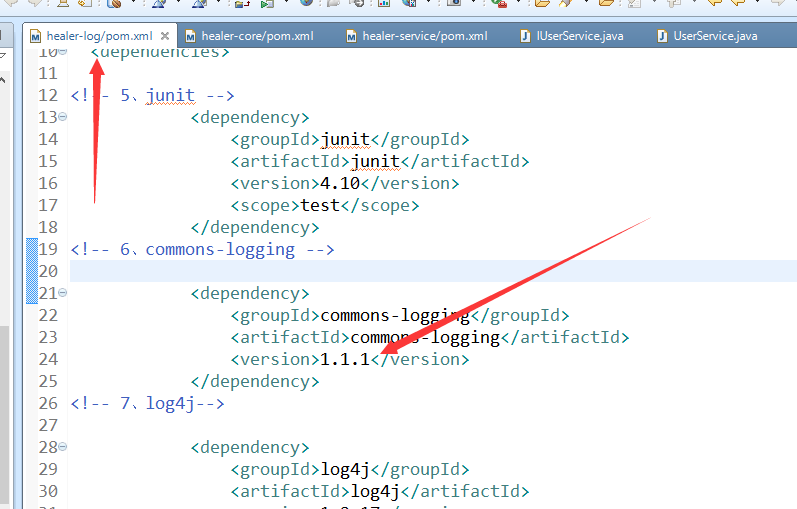
#### 3、如果是希望丢弃自身所引入的依赖，

#### 理解：下面的healer-core中的common-logging中的级别是由dbunit来的所以级别低，不论healer-core或者healer-log谁在前都是试用的healer-log中的版本

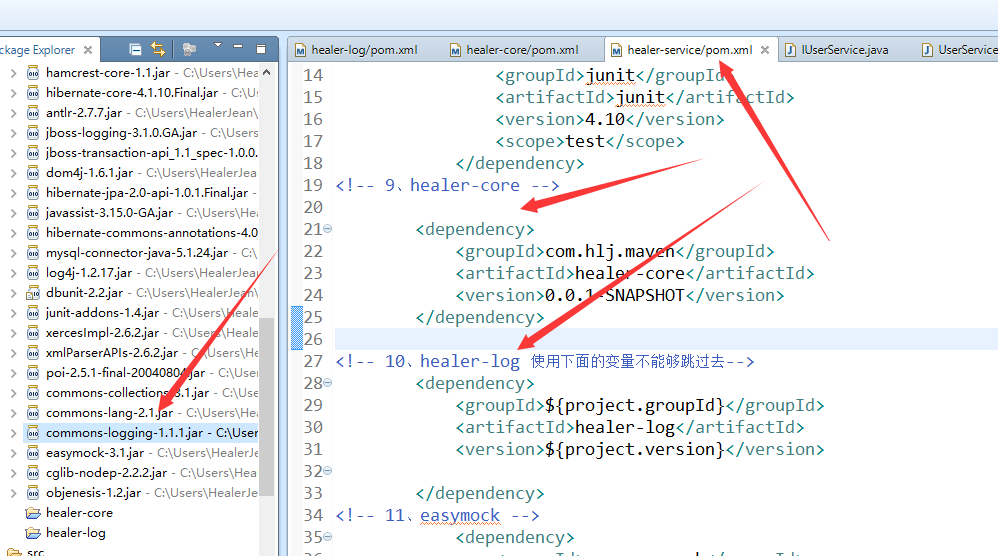
#### 1、healer-core中的common-logging是从这里dbunit继承而来



#### 2、healer-log中commons-logging 1.1.1

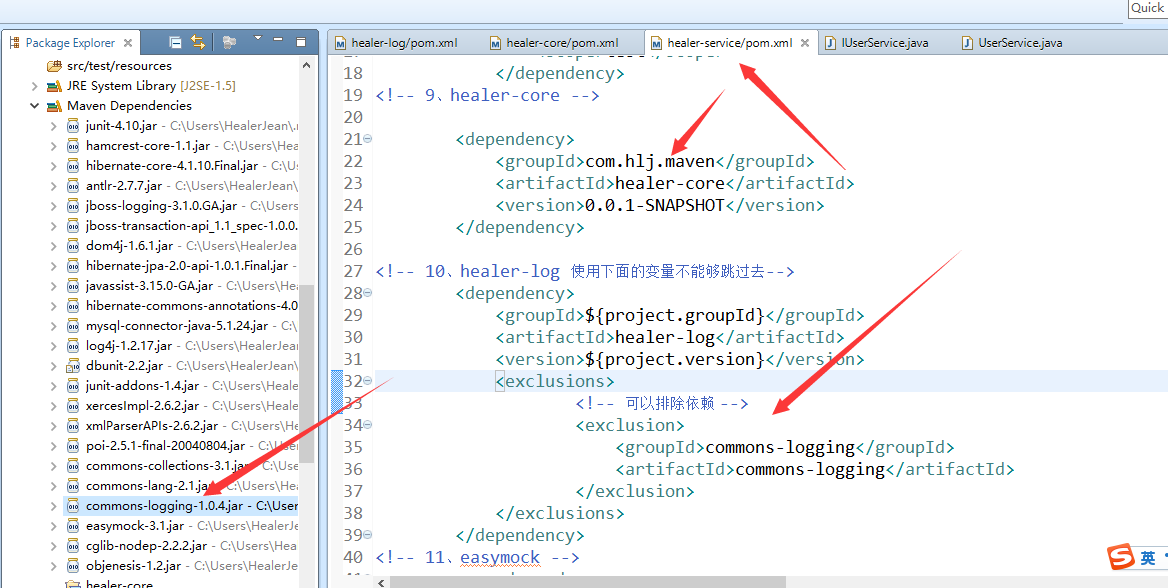


#### 3、healer-service 中的依赖版本为1.1.1 （不论他俩谁在前谁在后）



#### 4、<exclusions>排除万难，使用级别低的版本

|  |
| --- |
| <dependency>  <groupId>${project.groupId}</groupId>  <artifactId>healer-log</artifactId>  <version>${project.version}</version>  <exclusions>  <!-- 可以排除依赖 -->  <exclusion>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  </exclusion>  </exclusions>  </dependency> |



# 2、项目使用dbunit 不会对数据库真实数据产生操作，只是模拟而已

# 3、依赖参数

## 1.、scope 以mysql举例，只有测试和运行的时候才会依赖，所以这里设置为runtime

## 1、mysql scope为runtime

|  |
| --- |
| <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.24</version>  <scope>runtime</scope>  </dependency> |

## 2、junit scope test ，只有在编译test时候才会依赖

|  |
| --- |
| <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.10</version>  <scope>test</scope>  </dependency> |

