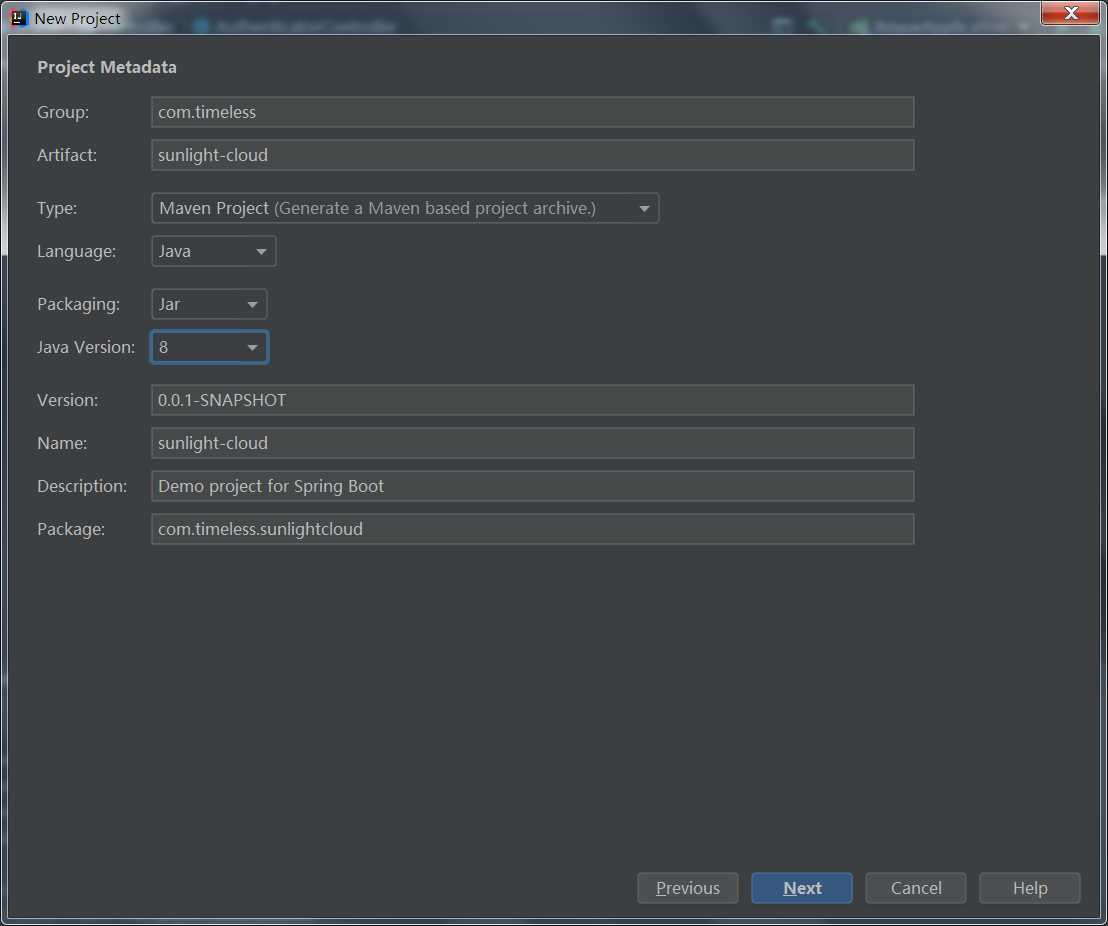
1.新建一个spring initial project维护所有模块pom



Pom文件中

<packaging>pom</packaging>

<modules>  
 <module>sunlight-common</module>  
 <module>sunlight-config</module>  
 <module>sunlight-eureka</module>  
 <module>sunlight-gateway</module>  
 <module>sunlight-api</module>  
 <module>sunlight-service</module>  
 <module>sunlight-monitor</module>  
</modules>

1. project下面新建各个module

Pom文件集成project

<parent>  
 <groupId>com.timeless</groupId>  
 <artifactId>sunlight-cloud</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</parent>

3.module下面还可以继续新建module 新建的时候注意维护层级 默认还是并列的module

Maven相关知识点

## **dependencyManagement使用简介**

Maven中的dependencyManagement元素提供了一种管理依赖版本号的方式。在dependencyManagement元素中声明所依赖的jar包的版本号等信息，那么所有子项目再次引入此依赖jar包时则无需显式的列出版本号。Maven会沿着父子层级向上寻找拥有dependencyManagement 元素的项目，然后使用它指定的版本号。

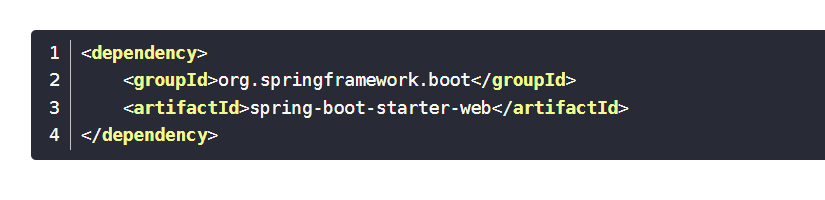
## **举例**

在父项目的POM.xml中配置：



此配置即生命了spring-boot的版本信息。

子项目则无需指定版本信息：



## **使用优点**

如果有多个子项目都引用同一样依赖，则可以避免在每个使用的子项目里都声明一个版本号。当想升级或切换到另一个版本时，只需要在顶层父容器里更新，而不需要逐个修改子项目；另外如果某个子项目需要另外的一个版本，只需要声明version即可。

## **注意事项**

dependencyManagement中定义的只是依赖的声明，并不实现引入，因此子项目需要显式的声明需要用的依赖。

## [dependencies与dependencyManagement的区别](https://www.cnblogs.com/feibazhf/p/7886617.html)

# **1、DepencyManagement应用场景**

         当我们的项目模块很多的时候，我们使用Maven管理项目非常方便，帮助我们管理构建、文档、报告、依赖、scms、发布、分发的方法。可以方便的编译代码、进行依赖管理、管理二进制库等等。

         由于我们的模块很多，所以我们又抽象了一层，抽出一个itoo-base-parent来管理子项目的公共的依赖。为了项目的正确运行，必须让所有的子项目使用依赖项的统一版本，必须确保应用的各个项目的依赖项和版本一致，才能保证测试的和发布的是相同的结果。

        在我们项目顶层的POM文件中，我们会看到dependencyManagement元素。通过它元素来管理jar包的版本，让子项目中引用一个依赖而不用显示的列出版本号。Maven会沿着父子层次向上走，直到找到一个拥有dependencyManagement元素的项目，然后它就会使用在这个dependencyManagement元素中指定的版本号。

这样做的好处：统一管理项目的版本号，确保应用的各个项目的依赖和版本一致，才能保证测试的和发布的是相同的成果，因此，在顶层pom中定义共同的依赖关系。同时可以避免在每个使用的子项目中都声明一个版本号，这样想升级或者切换到另一个版本时，只需要在父类容器里更新，不需要任何一个子项目的修改；如果某个子项目需要另外一个版本号时，只需要在dependencies中声明一个版本号即可。子类就会使用子类声明的版本号，不继承于父类版本号。

**总结区别：**

         <dependencies>中的jar直接加到项目中，管理的是依赖关系（如果有父pom,子pom,则子pom中只能被动接受父类的版本）；<dependencyManagement>主要管理版本，对于子类继承同一个父类是很有用的，集中管理依赖版本不添加依赖关系，对于其中定义的版本，子pom不一定要继承父pom所定义的版本。

项目的打包类型：pom、jar、war

项目中一般使用maven进行模块管理，每个模块下对应都有一个pom文件，pom文件中维护了各模块之间的依赖和继承关系。项目模块化可以将通用的部分抽离出来，方便重用；修改一部分代码不再是build整个项目，缩短了build时间；此外各模块都有自己的pom文件，结构更清晰。

使用maven进行模块划分管理，一般都会有一个父级项目，pom文件除了GAV(groupId, artifactId, version)是必须要配置的，另一个重要的属性就是packaging打包类型，所有的父级项目的packaging都为pom，packaging默认是jar类型，如果不作配置，maven会将该项目打成jar包。作为父级项目，还有一个重要的属性，那就是modules，通过modules标签将项目的所有子项目引用进来，在build父级项目时，会根据子模块的相互依赖关系整理一个build顺序，然后依次build。

而对于各个子项目，需要在其对应的pom文件开头申明对父级项目的引用，通过GAV实现。对于子项目自己的GAV配置，GV如果不配置，则会从父级项目的配置继承过来。子模块可通过dependencies标签来添加自己的依赖，此外子类项目的packaging值只能是war或者jar，前面已经说过，packaging默认是jar类型。如果是需要部署的项目，则需要打包成war类型，如果只是内部调用或者是作服务使用，则推荐打包成jar类型。

****其中子模块如果非代码层可以是packaging可以是pom包含多层pom,最终后为jar或war包****

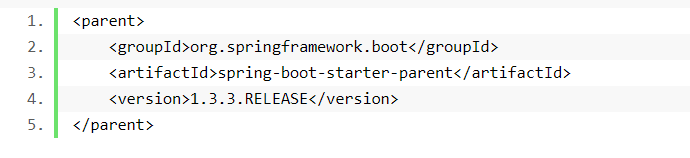
<packaging>pom</packaging>项目里没有java代码，也不执行任何代码，只是为了聚合工程或传递依赖用的。

**<type>**pom**</type>**

**<scope>**import**</scope>**

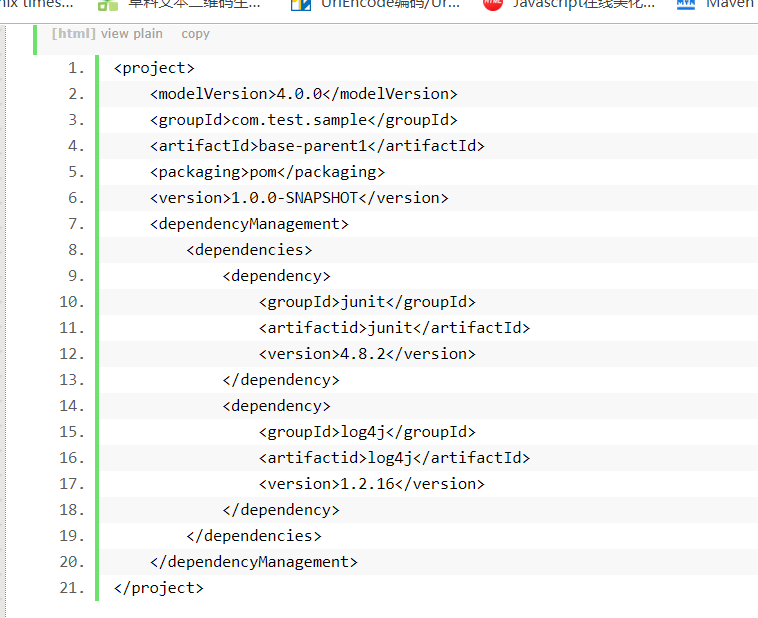
****以上配合的含义解释****

****常规可以这么引入父pom****



还可以

父类pom



这样引入



1. <type>pom</type>
2. <scope>import</scope>

必须写在dependencyManagement标签中

更详细的解释



