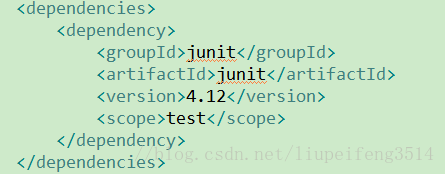
Maven是核心作用之一是为我们管理项目的依赖，为了能够准确的定位到每一个Java构件，每个构件都有一个唯一标识，这是Maven为我们管理项目的基础。

# Maven坐标

在平面几何或立体几何中可以用坐标来唯一标识一个点，为了准确的定位到某个Java构件，Maven制定了一套规则对构件进行标识。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。



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。

**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。