# Archetype

## Archetype使用再述：

### Maven Archetype Plugin：

Archetype并不是Maven的核心特性，它是通过插件来实现的，这一插件就是maven-archetype-plugin。尽管它只是一个插件，但由于其使用范围非常广泛，主要的IDE在集成Maven的时候都着重集成了archetype特性，以便用户快速的创建Maven项目。

maven-archetype-plugin插件1.x和2.x版本差异很大。在1.x版本中，使用Archetype创建项目使用的目标是archetype:create，但这一目标在2.x版本中已经不推荐使用，取而代之的是archetype:generate。它们主要的差异在于，前者要求用户必须一次性的从命令行输入所有的插件参数，而后者默认使用交互的方式提示用户选择或输入参数。不仅如此，archetype:generate也完全支持archetype:create的特性，因此用户已经完全没有必要去使用旧的archetype:create目标了。

### 1.1.2 使用archetype的一般步骤：

之前推荐用户在使用Archetype插件的时候输入完全的插件坐标，以防止Maven下载最新的不稳定快照版本。然而这种情况只是对于Maven2用户存在，在Maven3中，如果插件的版本未声明，Maven只会自动解析最新的发布版，因此用户不必担心引入快照版本带来的问题。以下是两条命令的对比：

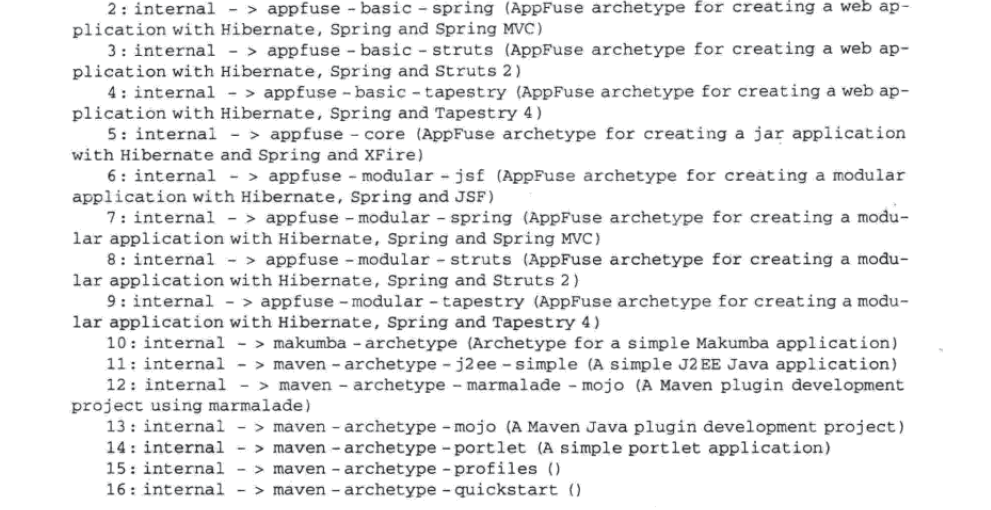
【Maven3】

|  |
| --- |
| mvn archetype:generate |

【Maven2】

|  |
| --- |
| mvn org.apache.maven.plugins:maven-archetype-plugin:2.0-alpha-5:generate |

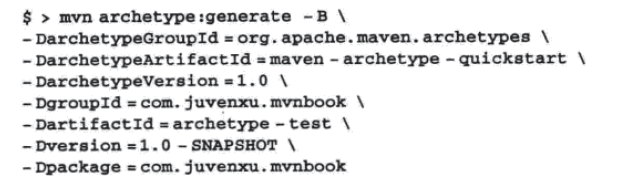
输入上述命令后，Archetype插件会输出一个Archetype列表供用户选择。例如：



这个列表来自于名为archetype-catalog.xml的文件。

### 1.1.3 批处理方式使用Archetype：

有时候用户可能不希望以交互的方式使用Archetype，例如当创建Maven项目的命令在一段自动化的Shell脚本中的时候，交互的方式会破坏自动化。这时用户可以使用mvn命令的-B选项，要求maven-archetype-plugin以批处理的方式运行。不过，这时用户还必须显式的声明要使用的Archetype坐标信息，以及要创建项目的groupId、artifactId、version、packaging等信息：



另外还有一种创建的方式，在之前的小节【[002--Maven\_\_使用骨架(模板)创建Maven项目](002--Maven__使用骨架(模板)创建Maven项目.docx)】已经介绍过。

### 1.1.4 常用Archetype介绍：

#### 1.1.4.1 maven-archetype-quickstart：

maven-archetype-quickstart可能是最常用的Archetype，当maven-archetype-plugin提示用户选择Archetype的时候，它就是默认值。使用maven-archetype-quickstart生成的项目十分简单，基本内容如下：

* 一个包含JUnit依赖声明的pom.xml
* src/main/java主代码目录及该目录下一个名为App的输出是“Hello World！”的类。
* src/test/java 测试代码目录及该目下一个名为AppTest的JUnit测试用例。

当需要创建一个全新的Maven项目时，基于可以使用该Archetype生成项目后进行修改，省去手工创建POM及目录结构的麻烦。

#### 1.1.4.2 maven-archetype-webapp：

这是一个简单的Maven War项目模板，当需要创建一个Web应用的时候就可以使用它。使用maven-archetype-webapp生成的项目内容如下：

* 一个packaing为war且带有JUnit依赖声明的pom.xml。
* src/main/webapp/目录
* src/main/webapp/index.jsp文件，一个简单的Hello World页面。
* src/main/webapp/WEB-INF/web.xml文件，一个基本为空的Web应用配置文件。

#### 1.1.4.3 AppFuse Archetype：

AppFuse是一个集成很多开源工具的项目，它由Matt Raible开发，旨在帮助Java编程人员快速高效的创建项目。AppFuse本身使用Maven构建，它的核心其实就是一个项目的骨架，是包含持久层、业务层及展现层的一个基本结构。在AppFuse2.x中，已经集成了大量流行的开源工具，如Spring、Structs2、JPA、JSF、Tapestry等。

AppFuse为用户提供大量的Archetype，以方便用户快速创建各种类型的项目，它们都使用同样的groupId org.appfuse。针对各种展现层框架分别为：

* appfuse-\*-jsf：基于JSF展现层框架的Archetype。
* appfuse-\*-spring：基于SpringMVC展现层框架的Archetype。
* appfuse-\*-structs：基于Structs2展现层框架的Archetype。
* appfuse-\*-tapestry：基于Tapestry展现层框架的Archetype。

每一种展现层框架都有3个Archetype，分别为light、basic和modular。其中，light类型的Archetype只包含最简单的骨架；basic类型的Archetype则包含了一些用户管理及安全方面的特性；modular类型的Archetype会生成多模块的项目，其中core模块包含了持久层及业务层的代码，而Web模块则是展现层的代码。

## 编写Archetype：

也许你所在的组织的一些项目都使用同样的框架和项目结构，为一个个项目重复同样的配置及同样的目录结构显然是难以让人接受的。更好的做法是创建一个属于自己的Archetype，这个Archetype包含了一些通用的POM配置、目录结构，甚至是Java类及资源文件，然后在创建项目的时候，就可以直接使用该Archetype，并提供一些基本参数，如groupId、artifactId、version，maven-archetype-plugin会处理其他原本需要手工处理的劳动。这样不仅节省了时间，也降低了错误配置发生的概率。

下面介绍一下创建Archetype的样例。首先读者需要了解的是，一个典型的Archetype Maven项目主要包含如下几个部分：

1. pom.xml：

Archetype自身的POM

1. src/main/resources/archetype-resources/pom.xml：

基于该Archetype生成的项目的POM原型

1. src/main/resources/MATE-INF/maven/archetype-metadata.xml：

Archetype的描述文件

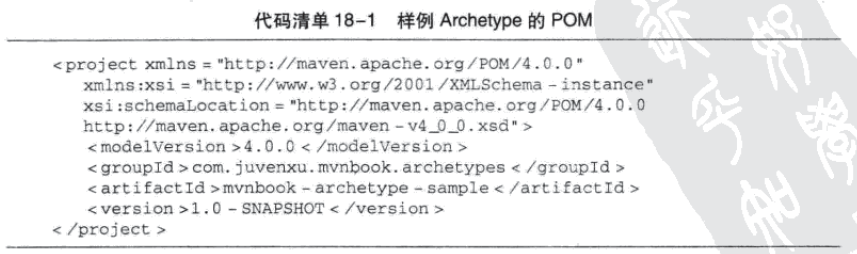
1. src/main/resources/archetype-resources/\*\*：

其他需要包含在Archetype中的内容

下面结合样例对上述内容一一详细解释。

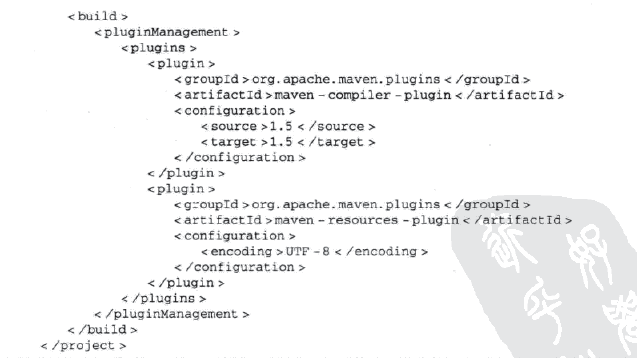
首先，和任何其他Maven项目一样，Archetype项目自身也需要一个POM。这个POM主要包含Archetype的坐标信息，这样Maven才能定位并使用它。读者还要留意，不要混淆Archetype的坐标和使用该Archetype生成的项目的坐标。需要主要的是，虽然Archetype可以说是一种特殊的Maven项目，但maven-archetype-plugin并没有要求Archetype项目使用特殊的打包类型。因此，一般来说，Archetype的打包类型就是默认值jar。

代码清单18-1展示了一个很简单的Archetype的POM：



接下来要关注的就是Archetype所包含的项目骨架的信息。从本质上来说，在编写Archetype的时候预先定义好其要包含的目录结构和文件，同时在必要的地方使用可配置属性声明代替硬编码。例如，项目的坐标信息一般都是可配置的。代码清单18-2就是一个简单的POM原型，它位于Archetype项目资源目录下的archetype-resources/子目录中。





【注】上面两个截图是一个POM

上述片段中的groupId、artifactId和version等信息并没有直接声明，而是使用了属性声明。

上述POM原型中还包含了一个JUnit依赖声明和两个插件配置。事实上，我们可以根据自己的实际需要在这里提供任何合法的POM配置，在使用该Archetype生成项目的时候，这些配置就是现成的。

一个Archetype最核心的部分是archetype-metadata.xml描述符文件，它位于Archetype项目资源目录的META-INF/maven子目录下。它主要用来控制两件事：

1. 声明哪些目录及文件应该包含在Archetype中；
2. 这个Archetype使用哪些属性参数；

代码清单18-3展示了一个Archetype描述符文件：

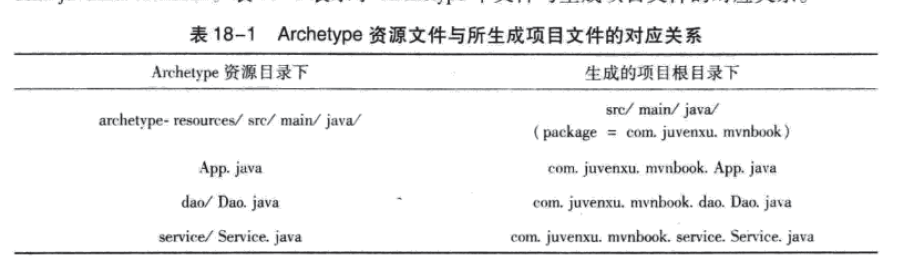


该例中Archetype描述符定义了名称为sample。它主要包含了fileSets和requireProperties两个部分。其中，fileSets可以包含一个或者多个fileSet子元素，每个fileSet定义一个目录，以及与该目录相关的包含或排除规则。

上述代码片段中的第一个fileSets指向的目录是src/main/java，该目录对应于Archetype项目资源目录的archetype-resources/src/main/java子目录。该fileSet有两个属性，filtered表示是否对该文件集合应用属性替换。例如，像${x}这样的内容是否替换为命令行输入的x参数的值；packaged表示是否将该目录下的内容放到生成项目的包路径下。

该fileSet还包含了includes子元素，并且声明了一个值为\*\*/\*.java的include规则，表示包含src/main/java/中任意路径下的java文件。这里两个\*号表示匹配任意目录，一个\*号表示匹配除路径分隔符外的任意0个或多个字符。这种匹配声明的方式在Maven的很多插件中都被用到。除了includes，用户还可以使用excludes声明要排除的文件。配置方法与includes类似，这里就不再赘述。

为了能说明问题，这里在src/main/resources/archetype-resources/src/main/java/目录下创建了一些文件，假设使用该Archetype创建项目的时候，package参数的值为com.juvenxu.mvnbook。表18-1表示了Archetype中文件与生成项目文件的对应关系：



如果fileSet的packaged属性值为true，directory的值为X，那么archetype-resources下的X目录就会对应在生成的项目中创建，在生成项目的该X目录下还会生成一个包目录，如上例中的com/juvenxu/mvnbook/，最后Archetype中X目录的子目录及文件都被复制到生成项目X目录的包目录下。如果packaged属性值为false，那么Archetype中X目录下的内容会被直接复制到生成项目的X目录下。

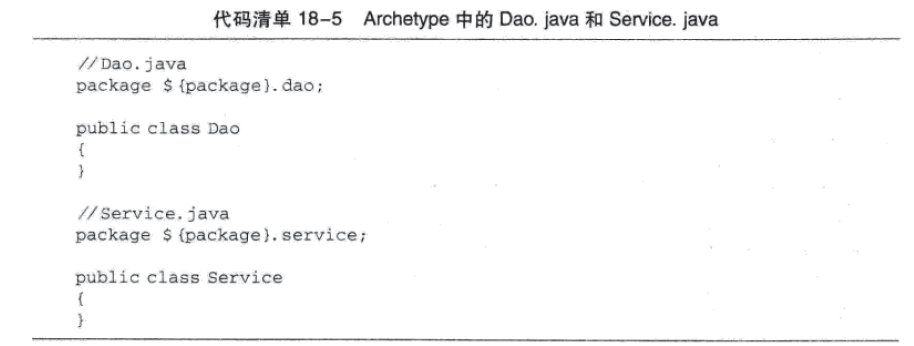
一般来说，java代码都需要放到包路径下，而项目资源文件则不需要。因此，在代码清单18-3中，第一个、第二个对应Java文件的fileSet的packaged的属性为true，而第三个对应资源文件的fileSet的packaged属性为false。

还有一点需要解释的是fileSet的filtered属性，它表示使用参数值替换属性声明，这是个非常有用的特性。例如，表18-1中涉及的几个Java类都需要有package声明，而且其值是在项目生成的时候确定的。这时就可以在Java代码中使用属性声明，如App.java的内容应该如代码清单18-4所示：



在使用报名com.juvenxu.mvnbook创建项目后，上述代码中的第一行会变成package com.juvenxu.mvnbook。

类似的，Dao.java和Service.java的包声明应该如代码清单18-5所示：



默认情况下，maven-archetype-plugin要求用户在使用Archetype生成项目的时候必须提供4个参数：groupId、artifactId、version和package。除此之外，用户编写Archetype的时候可以要求额外的参数。例如，代码清单18-3就使用了requireProperties配置要求额外的port参数，这样，Archetype中所有开发filtered的文件中就可以使用${port}属性声明，然后在项目生成的时候用命令行输入的值填充。

此外，编写Archetype的时候还可以为预置的4个参数提供默认值。例如，代码清单18-3中就为groupId参数提供了默认值com.juvenxu.mvnbook。在组织内部，可能很多项目的groupId是确定的，这时就可以为Archetype提供默认的groupId。

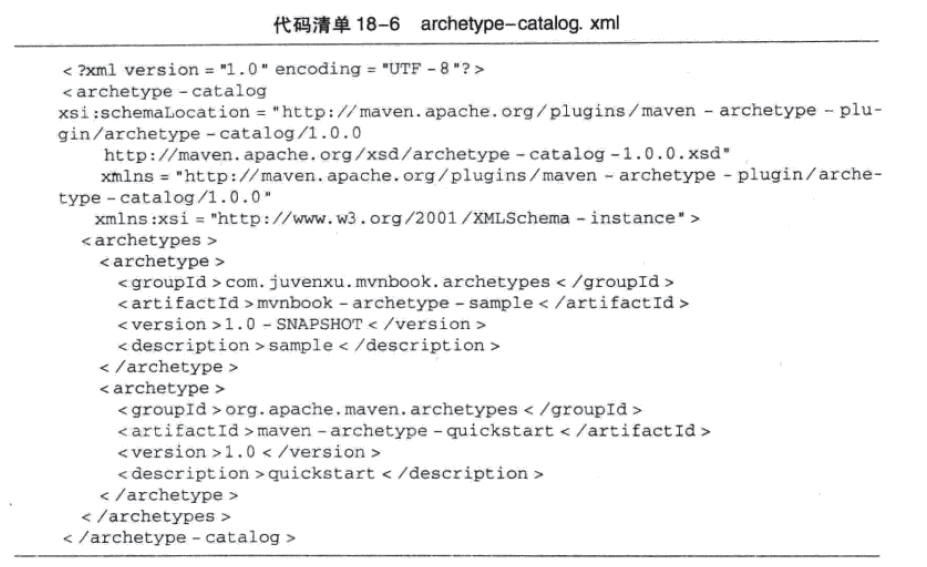
Archetype编写完成后，使用mvn clean install将其安装到本地仓库。接着用户就可以通过指定该Archetype的坐标用它生成项目。

## Archetype Catalog：

maven-archetype-plugin会提供一个Archetype列表供我们选择，那么，我们能够将自己创建的Archetype加入到这个列表中呢？答案是肯定的。下面就介绍相关的做法及相关原理。

### 1.3.1 什么是Archetype Catalog：

当用户以不指定Archetype坐标的方式使用maven-archetype-plugin的时候，会得到一个Archetype列表供选择，这个列表的信息来源于一个名为archetype-catalog.xml的文件。例如，代码清单18-6是一个包含两个Archetype信息的archetype-catalog.xml文件：



上述archetype-catalog.xml包含的两个Archetype读者应该已经熟悉了，第一个Archetype的坐标是com.juvenxu.mvnbook.archetypes:mvnbook-archetype-sample:1.0-SNAPSHOT，也就是上一节自定义的Archetype；第二个则是maven-archetype-plugin默认使用的Quickstarct Archetype。这个XML非常简单，它主要包含了各个Archetype的坐标。这样，当用户选择使用某个Archetype的时候，Maven就能立刻定位到Archetype构件。

### 1.3.2 Archetype Catalog的来源：

archetype-catalog.xml能够提供Archetype的信息，那么maven-archetype-plugin可以从哪些位置读取archetype-catalog.xml文件呢？下面是一个列表：

* internal：

这是maven-archetype-plugin内置的Archetype Catalog，包含了约58个Archetype信息。

* local：

指向用户本地的Archetype Catalog，其位置为~/.m2/archetype-catalog.xml。需要注意的是，该文件默认是不存在的。

* remote：

指向Maven中央仓库的Archetype Catalog，其确切地址为http://repo1.maven.org/maven2/archetype-catalog.xml。

* file://...：

用户可以指定本机任何位置 archetype-catalog.xml文件。

* http://...：

用户可以使用HTTP协议指定远程的archetype-catalog.xml文件。

当用户运行mvn archetype:generate 命令的时候，可以使用archetypeCatalog参数指定插件使用的Catalog。例如：



上述命令指定Archetype插件使用系统/tmp目录下的archetype-catalog.xml文件。当然，用户不需要每次运行Archetype目标的时候都去指定Catalog。在maven-archetyep-plugin 2.0-beta-4之前的版本中，archetypeCatalog的默认值是“internal,local”，即默认使用插件内置加上用户本机的Catalog信息，而从maven-archetype-plugin 2.0-beta-5开始，这一默认值变成了“remote,local”即默认使用中央仓库加上本机的Catalog信息。用户也可以使用逗号分隔多个Catalog来源。例如：

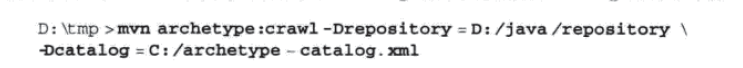


### 1.3.3 生成本地仓库的Archetype Catalog：

maven-archetype-plugin提供了一个名为crawl的目标，用户可以用它来遍历本地Maven仓库的内容名并自动生成archetype-catalog.xml文件，例如：



如果不提供任何参数，crawl目标会遍历用户settings.xml定义的localRepository，并且在该仓库的根目录下生成archetype-catalog.xml文件。用户可以使用参数repository指定要遍历的Maven仓库，使用参数catalog指定要更新的catalog文件。例如：



将自定义的Archetype安装到本地之后，接着用户就可以在创建项目的时候指定使用该Catalog。

### 1.3.4 使用nexus-archetype-plugin：

Nexus团队提供了一个名为nexus-archetype-plugin的插件，该插件能够基于Nexus仓库时实的生成archetype-catalog.xml文件。由于catalog内容是基于仓库索引生成而不是逐个遍历仓库文件，因此生成的速度非常快。只要用户安装了该插件，每个Nexus仓库都会随时提供一个与索引内容一致的Catalog。

用户可以从以下地址下载最新的nexus-archetype-plugin：<http://repository.sonatype.org/content/groups/forge/org/sonatype/nexus/plugins/nexus-archetype-plugin/>

下一步是将nexus-archetype-plugin插件的bundle.zip包解压到Nexus工作目录sonatype-work/nexus/下的plugin-repository/子目录中，然后重启Nexus，插件就安装完成了。

现在，当用户浏览Nexus仓库内容的时候，就能够在仓库的根目录下看到archetype-catalog.xml文件，右击选择“Download”后就能够下载该文件。