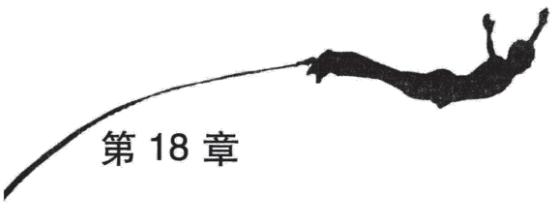

第 18 章

Archetype

本章内容

- ❑ Archetype 使用再叙
- ❑ 编写 Archetype
- ❑ Archetype Catalog
- ❑ 小结

3.5 节已经简单介绍了如何使用 Maven Archetype 快速生成项目骨架。读者可以将 Archetype 理解成 Maven 项目的模板，例如 maven-archetype-quickstart 就是最简单的 Maven 项目模板，只需要提供基本的元素（如 groupId、artifactId 及 version 等），它就能生成项目的基本结构及 POM 文件。很多著名的开源项目（如 AppFuse 和 Apache Wicket）都提供了 Archetype 方便用户快速创建项目。如果你所在组织的项目都遵循一些通用的配置及结构，则也可以为其创建一个自己的 Archetype 并进行维护。使用 Archetype 不仅能让用户快速简单地创建项目；还可以鼓励大家遵循一些项目结构及配置约定。

18.1 Archetype 使用再叙

3.5 节已经介绍了 Archetype 的基本使用方法，本节进一步解释相关原理及一些常用的 Archetype。

18.1.1 Maven Archetype Plugin

Archetype 并不是 Maven 的核心特性，它也是通过插件来实现的，这一插件就是 maven-archetype-plugin（http://maven.apache.org/archetype/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（如 Eclipse、NetBeans 和 IDEA）在集成 Maven 的时候都着重集成了 archetype 特性，以方便用户快速地创建 Maven 项目。

在本书编写的时候，maven-archetype-plugin 最新的版本是 2.0-alpha-5。需要特别注意的是，该插件的 1.x 版本和 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了，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地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。

18.1.2 使用 Archetype 的一般步骤

3.5 节推荐用户在使用 Archetype 插件的时候输入完整的插件坐标，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 Maven 2 用户存在，在 Maven 3 中，如果插件的版本未声明，Maven 只会自动解析最新的发布版，因此用户不用担心引入快照版本带来的问题。以下是两条命令的对比：

- **Maven 3**：mvn archetype：generate
- **Maven 2**：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：

```
Choose archetype:
1: internal -> appfuse-basic-jsf (AppFuse archetype for creating a web application with Hibernate, Spring and JSF)
```

```
2: internal -> appfuse-basic-spring (AppFuse archetype for creating a web application with Hibernate, Spring and Spring MVC)
3: internal -> appfuse-basic-struts (AppFuse archetype for creating a web application with Hibernate, Spring and Struts 2)
4: internal -> appfuse-basic-tapestry (AppFuse archetype for creating a web application with Hibernate, Spring and Tapestry 4)
5: internal -> appfuse-core (AppFuse archetype for creating a jar application with Hibernate and Spring and XFire)
6: internal -> appfuse-modular-jsf (AppFuse archetype for creating a modular application with Hibernate, Spring and JSF)
7: internal -> appfuse-modular-spring (AppFuse archetype for creating a modular application with Hibernate, Spring and Spring MVC)
8: internal -> appfuse-modular-struts (AppFuse archetype for creating a modular application with Hibernate, Spring and Struts 2)
9: internal -> appfuse-modular-tapestry (AppFuse archetype for creating a modular application with Hibernate, Spring and Tapestry 4)
10: internal -> makumba-archetype (Archetype for a simple Makumba application)
11: internal -> maven-archetype-j2ee-simple (A simple J2EE Java application)
12: internal -> maven-archetype-marmalade-mojo (A Maven plugin development project using marmalade)
13: internal -> maven-archetype-mojo (A Maven Java plugin development project)
14: internal -> maven-archetype-portlet (A simple portlet application)
15: internal -> maven-archetype-profiles ()
16: internal -> maven-archetype-quickstart ()
...
```

这个列表来自于名为 archetype-catalog.xml 的文件，18.3 节将对其进行深入解释。现在，用户需要选择自己想要使用的 Archetype，然后输入其对应的编号。

由于 Archetype 只是一个模板，为了保持模板的通用性，它的很多重要内容都是可配置的。因此，在用户选择了一个 Archetype 之后，下一步就需要提供一些基本的参数。主要有：

- **groupId**：想要创建项目的 groupId。
- **artifactId**：想要创建项目的 artifactId。
- **version**：想要创建项目的 version。
- **package**：想要创建项目的默认 Java 包名。

上述参数是 Archetype 插件内置的，也是最常用和最基本的。用户在自己编写 Archetype 的时候，还可以声明额外的配置参数。

根据 Maven 提示填写完配置参数之后，Archetype 插件就能够生成项目的骨架了。

18.1.3 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地声明要使用的 Archetype 坐标信息，以及要创建项目的 groupId、artifactId、version、package 等信息。例如：

```
$ > mvn archetype:generate -B \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DarchetypeVersion=1.0 \
-DgroupId=com.juvenxu.mvnbook \
-DartifactId=archetype-test \
-Dversion=1.0-SNAPSHOT \
-Dpackage=com.juvenxu.mvnbook
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] > > > maven-archetype-plugin:2.0-alpha-5:generate (default-cli) @
standalone-pom > > >
[INFO]
[INFO] < < < maven-archetype-plugin:2.0-alpha-5:generate (default-cli) @
standalone-pom < < <
[INFO]
[INFO] ---maven-archetype-plugin:2.0-alpha-5:generate (default-cli) @ stan-
dalone-pom ---
[INFO] Generating project in Batch mode
[INFO] Archetype repository missing. Using the one from [org.apache.maven.arche-
types:maven-archetype-quickstart:1.0] found in catalog remote
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating OldArchetype: maven-archetype-
quickstart:1.0
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.juvenxu.mvnbook
[INFO] Parameter: packageName, Value: com.juvenxu.mvnbook
[INFO] Parameter: package, Value: com.juvenxu.mvnbook
[INFO] Parameter: artifactId, Value: archetype-test
[INFO] Parameter: basedir, Value: D:\tmp\archetype
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] * * * * * * * * * * * * * * * * * * * * * * * * End of debug info from resources from
generated POM * * * * * * * * * * * * * * * * * * * * * * * *
[INFO] OldArchetype created in dir: D:\tmp\archetype\archetype-test
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.624s
[INFO] Finished at: Wed Apr 28 14:34:32 CST 2010
[INFO] Final Memory: 6M/11M
[INFO] ------------------------------------------------------------------------
```

该例中的 Archetype 的坐标为 org. apache. maven. archetypes：maven- archetype- quckstart：1.0，而真正要创建的项目坐标则为 com. juvenxu. mvnbook：archetype- test：1. 0- SNAPSHOT。

18. 1. 4 常用 Archetype 介绍

在编写本书的时候，Maven 中央仓库中已经包含了 249 个 Archetype（详见 http://repo1. maven. org/maven2/archetype-catalog. xml）。此外，还有大量没有发布到中央仓库的 Ar-

chetype 分布在其他 Maven 仓库中。任何人都不可能全部了解它们，因此这里只介绍几个比较常用的 Archetype。

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“Hello World!”的类。
- src/ test/ java 测试代码目录及该目录下一个名为 AppTest 的 JUnit 测试用例。

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

2. maven- archetype- webapp

这是一个最简单的 Maven war 项目模板，当需要快速创建一个 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 packaging 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。

3. AppFuse Archetype

AppFuse 是一个集成了很多开源工具的项目，它由 Matt Raible 开发，旨在帮助 Java 编程人员快速高效地创建项目。AppFuse 本身使用 Maven 构建，它的核心其实就是一个项目的骨架，是包含了持久层、业务层及展现层的一个基本结构。在 AppFuse 2. x 中，已经集成了大量流行的开源工具，如 Spring、Struts 2、JPA、JSF、Tapestry 等。

AppFuse 为用户提供了大量 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 Spring MVC 展现层框架的 Archetype。
- **appfuse- * - struts**：基于 Struts 2 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，其中的 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。

更多关于 AppFuse Archetype 的信息，读者可以访问其官方的快速入门手册：http：//appfuse. org/ display/ apf/ appfuse + quickstart。

18.2 编写 Archetype

也许你所在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。

下面就介绍一个创建 Archetype 的样例。首先读者需要了解的是，一个典型的 Archetype Maven 项目主要包括如下几个部分：

- **pom. xml**：Archetype 自身的 POM。
- **src /main /resources /archetype- resources /pom. xml**：基于该 Archetype 生成的项目的 POM 原型。
- **src /main /resources /META- INF /maven /archetype- metadata. xml**：Archetype 的描述符文件。
- **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 项目自身也需要有一个 POM。这个 POM 主要包含该 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。需要注意的是，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。

代码清单 18-1 样例 Archetype 的 POM

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance"
    xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven - v4_0_0.xsd" >
    <modelVersion >4.0.0 < /modelVersion >
    <groupId >com.juvenxu.mvnbook.archetypes < /groupId >
    <artifactId >mvnbook - archetype - sample < /artifactId >
    <version >1.0 - SNAPSHOT < /version >
< /project >
```

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的属

性声明替代硬编码。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。

代码清单 18-2 样例 Archetype 所包含的 POM 原型

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>${groupId}</groupId>
    <artifactId>${artifactId}</artifactId>
    <version>${version}</version>
    <name>${artifactId}</name>
    <url>http://www.juvenxu.com</url>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
                        <source>1.5</source>
                        <target>1.5</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-resources-plugin</artifactId>
                    <configuration>
                        <encoding>UTF-8</encoding>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

上述代码片段中的 groupId、artifactId 和 version 等信息并没有直接声明，而是使用了属性声明。回顾 18.1.2 节，使用 Archetype 生成项目的时候，用户一般都需要提供 groupId、artifactId、version、package 等参数，在那个时候，这些属性声明就会由那些参数值填充。

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了。

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情：一是声明哪些目录及文件应该包含在 Archetype 中；二是这个 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。

代码清单 18-3 样例 Archetype 描述符文件

```
<?xml version="1.0" encoding="UTF-8"?>
<archetype-descriptor name="sample">
    <fileSets>
        <fileSet filtered="true" packaged="true">
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.java</include>
            </includes>
        </fileSet>
        <fileSet filtered="true" packaged="true">
            <directory>src/test/java</directory>
            <includes>
                <include>**/*.java</include>
            </includes>
        </fileSet>
        <fileSet filtered="true" packaged="false">
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.properties</include>
            </includes>
        </fileSet>
    </fileSets>
    <requiredProperties>
        <requiredProperty key="port" />
        <requiredProperty key="groupId">
            <defaultValue>com.juvenxu.mvnbook</defaultValue>
        </requiredProperty>
    </requiredProperties>
</archetype-descriptor>
```

该例中的 Archetype 描述符定义了名称为 sample。它主要包含 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 fileSet 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。18.1.2 节提到使用 Archetype 必须提供的参数之一就是 package，即项目包名。如果读者暂时无法理解这两个属性的作用，不必着急，稍后通过实例来解释。

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。这里两个星号 * * 表示匹配任意目录，一个星号 * 表示匹配除路径分隔符外的任意 0 个或者多个字符。这种匹配声明的方式在 Maven 的很多插件中都被用到，如 10.5 节中的 maven- surefire- plugin。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，这里不再赘述。

为了能够说明问题，笔者在 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。

表 18-1 Archetype 资源文件与所生成项目文件的对应关系

Archetype 资源目录下	生成的项目根目录下
archetype- resources/ src/ main/ java/	src/ main/ java/ (package = com. juvenxu. mvnbook)
App. java	com. juvenxu. mvnbook. App. java
dao/ Dao. java	com. juvenxu. mvnbook. dao. Dao. java
service/ Service. java	com. juvenxu. mvnbook. service. Service. java

如果 fileSet 的 packaged 属性值为 true，directory 的值为 X，那么 archetype- resources 下的 X 目录就会对应地在生成的项目中被创建，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被复制到生成项目 X 目录的包目录下。如果 packaged 的属性值为 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、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。

代码清单 18-4 Archetype 中的 App. java

```
package ${package};

public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

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

com. juvenxu. mvnbook;。

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

代码清单 18-5 Archetype 中的 Dao. java 和 Service. java

```
//Dao.java
package ${package}.dao;

public class Dao
{
}

//Service.java
package ${package}.service;

public class Service
{
}
```

对应地，项目生成后 Dao. java 的第一行会成为“package com. juvenxu. mvnbook. dao;”，而 Service. java 的第一行会成为“package com. juvenxu. mvnbook. service;”。使用这样的技巧，就可以在 Archetype 中创建多层次的 Java 代码。

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，用户在编写 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 中所有开启 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，在编写 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 编写完成之后，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了：

```
$ > mvn archetype:generate \
-DarchetypeGroupId=com.juvenxu.mvnbook.archetypes \
-DarchetypeArtifactId=mvnbook-archetype-sample \
-DarchetypeVersion=1.0-SNAPSHOT
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] > > > maven-archetype-plugin:2.0-alpha-5:generate (default-cli) @ standa-
lone-pom > > >
[INFO]
[INFO] < < < maven-archetype-plugin:2.0-alpha-5:generate (default-cli) @ standa-
lone-pom < < <
```

```
[INFO]
[INFO] ---maven-archetype-plugin:2.0-alpha-5:generate (default-cli) @ standa-
lone-pom ---
[INFO] Generating project in Interactive mode
[WARNING] No archetype repository found. Falling back to central repository (ht-
tp://repo1.maven.org/maven2).
[WARNING] Use -DarchetypeRepository = <your repository > if archetype's reposito-
ry is elsewhere.
[INFO] snapshot com.juvenxu.mvnbook.archetypes:mvnbook-archetype-sample:1.0-
SNAPSHOT: checking for updates from mvnbook-archetype-sample-repo
[INFO] Using property: groupId = com.juvenxu.mvnbook
Define value for property 'artifactId': : test
Define value for property 'version': 1.0 - SNAPSHOT: :
[INFO] Using property: package = com.juvenxu.mvnbook
Define value for property 'port': : 8080
Confirm properties configuration:
groupId: com.juvenxu.mvnbook
artifactId: test
version: 1.0 - SNAPSHOT
package: com.juvenxu.mvnbook
port: 8080
 Y: : y
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 27.365s
[INFO] Finished at: Sun May 02 01:13:10 CST 2010
[INFO] Final Memory: 5M/10M
[INFO] ------------------------------------------------------------------------
```

该例使用了交互式的方式生成项目，由于该 Archetype 为 groupId 定义了默认值，用户就不再需要输入 groupId 的值了。此外，用户还不得不输入该 Archetype 额外定义的 port 参数的值。

18.3 Archetype Catalog

18.2 节中我们自定义了一个 Archetype，然后可以通过指定该 Archetype 的坐标在命令行用它创建项目原型。但是，18.1.2 节告诉我们，通常使用 Archetype 不需要精确地指定 Archetype 的坐标，maven-archetype-plugin 会提供一个 Archetype 列表供我们选择。那么，能否把自己创建的 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。

18.3.1 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 是一个包含了两个 Archetype 信息的 archetype-catalog.xml 文件。

代码清单 18-6 archetype-catalog. xml

```
<?xml version="1.0" encoding="UTF-8"?>
<archetype-catalog
xsi:schemaLocation="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-catalog/1.0.0
    http://maven.apache.org/xsd/archetype-catalog-1.0.0.xsd"
    xmlns="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-catalog/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <archetypes>
    <archetype>
      <groupId>com.juvenxu.mvnbook.archetypes</groupId>
      <artifactId>mvnbook-archetype-sample</artifactId>
      <version>1.0-SNAPSHOT</version>
      <description>sample</description>
    </archetype>
    <archetype>
      <groupId>org.apache.maven.archetypes</groupId>
      <artifactId>maven-archetype-quickstart</artifactId>
      <version>1.0</version>
      <description>quickstart</description>
    </archetype>
  </archetypes>
</archetype-catalog>
```

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 Quick-start 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 就能够立刻定位到 Archetype 构件。

18. 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**：指向了 Maven 中央仓库的 Archetype Catalog，其确切的地址为 http://repo1. maven. org/maven2/archetype-Catalog. xml。在本书编写的时候，该 Catalog 包含了约 249 个 Archetype 信息。
- **file://...**：用户可以指定本机任何位置的 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：

```
$> mvn archetype:generate -DarchetypeCatalog = file:// /tmp/archetype-catalog.xml
```

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-archetype- plugin 2. 0-beta-4 之前的版本中，archetypeCatalog 的默认值为“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用户本机的 Catalog 信息。用户也可以使用逗号分隔多个 Catalog 来源。例如：

```
$> mvn archetype:generate -DarchetypeCatalog = file:// /tmp/archetype - catalog.xml,local
```

该命令指定 Archetype 从两个位置读取 Catalog 信息。

archetype：generate 的输出也会告诉用户每一条 Archetype 信息的来源。例如：

```
1: local - > mvnbook - archetype - sample (sample)
2: local - > maven - archetype - mojo (plugin)
3: local - > maven - archetype - quickstart (quickstart)
4: local - > maven - archetype - webapp (webapp)
5: internal - > appfuse - basic - jsf (AppFuse archetype...
6: internal - > appfuse - basic - spring (AppFuse archetype...
7: internal - > appfuse - basic - struts (AppFuse archetype...
8: internal - > appfuse - basic - tapestry (AppFuse archetype ...
9: internal - > appfuse - core (AppFuse archetype...
...
```

上述输出片段告诉用户，archetype 1-4 来源于本机的 ~/. m2/ archetype-catalog. xml 文件，而 archetype 5-9 来源于 Archetype 插件内置的 archetype- catalog. xml 文件。

18. 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 仓库的内容并自动生成 archetype- catalog. xml 文件。例如：

```
D:\tmp >mvn archetype:crawl
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] ---maven - archetype - plugin:2.0 - alpha - 5:crawl (default - cli) @ standalone - pom ---
repository D:\java\repository
catalogFile null
[INFO] Scanning D:\java\repository\ant\ant\1.5.1\ant - 1.5.1 - sources.jar
[INFO] Scanning D:\java\repository\ant\ant\1.5.1\ant - 1.5.1.jar
[INFO] Scanning D:\java\repository\ant\ant\1.6\ant - 1.6.jar
[INFO] Scanning D:\java\repository\ant\ant\1.6.5\ant - 1.6.5.jar
...
```

```
...
[INFO] Scanning D:\java\repository\xpp3\xpp3_min\1.1.4c\xpp3_min-1.1.4c-
sources.jar
[INFO] Scanning D:\java\repository\xpp3\xpp3_min\1.1.4c\xpp3_min-1.1.4c.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 19.355s
[INFO] Finished at: Sun May 02 15:43:37 CST 2010
[INFO] Final Memory: 3M/8M
```

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：

```
D:\tmp>mvn archetype:crawl -Drepository=D:/java/repository \
-Dcatalog=C:/archetype-catalog.xml
```

将自定义的 Archetype 安装到本地仓库后，使用 Archetype：crawl 基于该仓库生成的 Catalog 就会包含该 Archetype 的信息，接着用户就可以在创建项目的时候指定使用该 Catalog。

18.3.4 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 仓库索引实时地生成 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”后就能下载该文件，如图 18-1 所示。

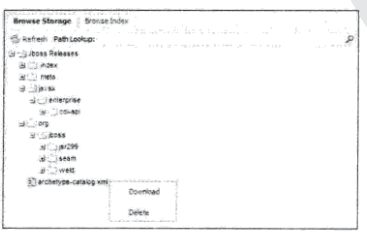

图 18-1 用 nexus- archetype- plugin 生成 Archetype Catalog

18.4 小结

本章详细阐述了最为有用的 Maven 插件之一：Maven Archetype Plugin。读者可以选择以交互式或者批处理的方式使用该插件生成项目骨架。另外，还介绍了一些常用的 Archetype。

本章的重点是教授读者创建自己的 Archetype，这主要包括理解 Archetype 项目的结构、如何通过属性过滤为 Archetype 提供灵活性，以及 Archetype Package 参数的作用。Archetype Plugin 通过读取 Archetype- catalog. xml 文件内容来提供可用的 Archetype 列表信息，这样的 Catalog 可以从各个地方获得，如插件内置、本机机器、中央仓库以及自定义的 file://或 http://路径。本章最后介绍了如何使用 archetype：crawl 和 nexus- archetype- plugin 生成仓库的 archetype- catalog. xml 内容。