项目管理: Maven 让事情变得简单

给您的下一个 **Java** 构建添加项目管理特性

developerWorks 中国

本文内容包括:

- Maven vs Ant
- Maven 的主要组件
- 安装 Maven
- 样本 J2EE 项目
- 参与其中
- 结束语
- 参考资料
- 关于作者
- 对本文的评价

相关链接:

- Java technology 技术文档库

文档选项

打印本页

将此页作为电子邮件发送

级别: 初级

Charles Chan (charlesc@ibiblio.org), 高级软件开发人员, Finetix LLC

2003 年 7 月 30 日

尽管 Ant 对于构建 Java 程序而言是事实上的标准工具，但这个工具在许多方面都不胜任项目管理任务。相反，Ant 提供的东西，Maven（出自 Apache Jakarta 项目的高级项目管理工具）都能提供，而且更多。Java 开发人员 Charles Chan 将介绍 Maven 的特性，并循序渐进地指导您进行一次完整的 Maven 项目设置。

目前，绝大多数开发人员都把 Ant 当作 Java 编程项目的标准构建工具。遗憾的是，Ant 的项目管理工具（作为 make 的替代工具）不能满足绝大多数开发人员的需要。通过检查 Ant 构建文件，很难发现项目的相关性信息和其它元信息（如开发人员／拥有者、版本或站点主页）。

Maven 除了以程序构建能力为特色之外，还提供 Ant 所缺少的高级项目管理工具。由于 Maven 的缺省构建规则有较高的可重用性，所以常常用两三行 Maven 构建脚本就可以构建简单的项目，而使用 Ant 则需要十几行。事实上，由于 Maven 的面向项目的方法，许多 Apache Jakarta 项目现在使用 Maven，而且公司项目采用 Maven 的比例在持续增长。

Maven vs Ant

那么，Maven 和 Ant 有什么不同呢？在回答这个问题以前，我要强调一点：Maven 和 Ant 针对构建问题的两个不同方面。Ant 为 Java 技术开发项目提供跨平台构建任务。Maven 本身描述项目的高级方面，它从 Ant 借用了绝大多数构建任务。因此，由于 Maven 和 Ant 代表两个差异很大的工具，所以我将只说明这两个工具的等同组件之间的区别，如表 1 所示。

表 1. Maven vs Ant

	Maven	**Ant**
标准构建文件	project.xml 和 maven.xml	build.xml
特性处理顺序	1. ${maven.home}/bin/driver.properties 2. ${project.home}/project.properties 3. ${project.home}/build.properties 4. ${user.home}/build.properties 5. 通过 -D 命令行选项定义的系统特性 最后一个定义起决定作用。	1. 通过 -D 命令行选项定义的系统特性 2. 由 <property> 任务装入的特性 第一个定义最先被处理。
构建规则	构建规则更为动态（类似于编程语言）；它们是基于 Jelly 的可执行 XML。	构建规则或多或少是静态的，除非使用 <script> 任务。（请参阅 参考资料以获得相关教程。）

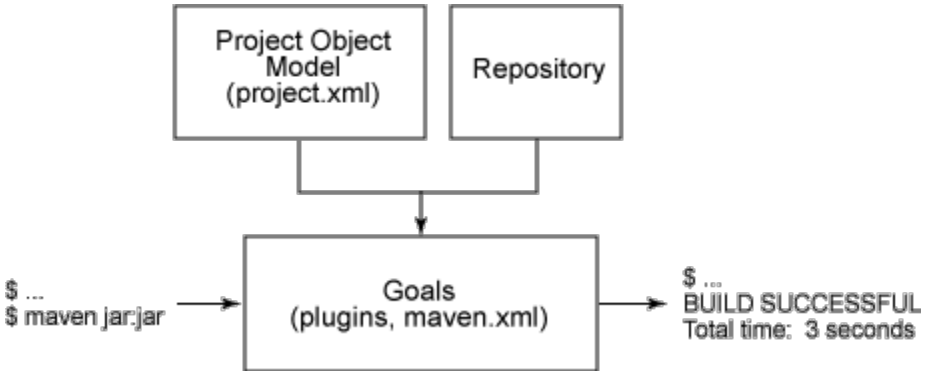
扩展语言	插件是用 Jelly（XML）编写的。	插件是用 Java 语言编写的。
构建规则可扩展性	通过定义 <preGoal> 和 <postGoal> 使构建 goal 可扩展。	构建规则不易扩展；可通过使用 <script> 任务模拟 <preGoal> 和 <postGoal> 所起的作用。

回页首

Maven 的主要组件

既然您了解了 Maven 和 Ant 之间的区别，让我们来研究 Maven 的主要组件，如图 1 所示。

图 **1. Maven** 的主要组件



项目对象模型

项目对象模型（Project Object Model，POM）描述项目的各个方面。尽管对于 POM 的物理表示没有内在的限制，但 Maven 开发人员通常使用一个 XML 项目文件（project.xml）。该 XML 文件格式由位于 Maven 安装目录中的 XML 模式（maven-project.xsd）定义。

通常，project.xml 文件由三个主要部分组成：

- 项目管理部分包括项目的组织、开发人员名单、源代码位置和错误跟踪系统 URL 等信息。
- 项目相关性部分包括关于项目相关性的信息。当前 Maven 实现（1.0 beta 测试版 8）仅支持 JAR 文件相关性。
- 项目构建和报告部分包含项目构建信息（如源代码目录、单元测试用例目录）和要在构建中生成的报告。

清单 1 显示了带注释的样本 project.xml 文件。因为 project.xml 文件中的许多元素都是可选的，所以，随着您对 Maven 理解的加深，可以逐步使用不同的 Maven 特性。 注：在以下代码中，可选的元素都以“可选的（*OPTIONAL*）”标明。

主文档包含项目的唯一标识和组标识。事实证明，当项目包括多个子项目时，组标识非常有用。所有的子项目应共享同一组标识，但每个子项目应有不同的 <id> 。

清单 **1.** 主 **project.xml** 框架

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- A project file's root element -->
```

```
<project>
  <!-- The POM version. This tag is currently unused. -->
  <pomVersion>3</pomVersion>
  <!-- A project group id. If present, the id serves as the project's
       directory name in the repository -->
  <groupId>crayola-group</groupId>
  <!-- A unique project identifier. The project identifier and its
       version number often generate file/directory names during the
       build. For example, a project JAR file follows the
       <id>-<version> naming convention. -->
  <id>crayola</id>
  <!-- A short name for the project -->
  <name>Crayola Professional</name>
  <!-- The project version number. Maven does not enforce a particular
       version numbering scheme. -->
  <currentVersion>0.0.1</currentVersion>
  ...
  <!--
------------------------------------------------------------------- -->
  <!-- Project management section                                   -->
  <!--
------------------------------------------------------------------- -->
  ...
  <!--
------------------------------------------------------------------- -->
  <!-- Project dependency section                                   -->
  <!--
------------------------------------------------------------------- -->
  ...
  <!--
------------------------------------------------------------------- -->
  <!-- Project build and reports section                            -->
  <!--
------------------------------------------------------------------- -->
  ...
</project>
```

项目管理部分（如清单 2 所示）主要包括可选项。在此部分中指定开发人员名单（带有正确的标识），当您希望获得更改日志（Change Log）报告和开发活动（Development Activity）报告时尤其要这么做。

清单 **2.** 项目管理部分

```
...
<!--
------------------------------------------------------------------- -->
  <!-- Project management section                                    -->
  <!--
------------------------------------------------------------------- -->
  <!-- Details of the organization holding the project. Only the name
        is required. -->
  <organization>
    <name>Markers Inc.</name>
    <url>http://w3.markers.com/</url>

<logo>http://w3.markers.com/logo/company-logo.gif</logo>
  </organization>
  <!-- (OPTIONAL) Year of inception -->
  <inceptionYear>2003</inceptionYear>
  <!-- (OPTIONAL) Project main package -->
  <package>com.markers.crayola.*</package>
  <!-- (OPTIONAL) Project logo picture (URL) -->
  <logo>http://w3.markers.com/logo/crayola.gif</logo>
  <!-- (OPTIONAL) GUMP repository ID. Useful only if you use GUMP. -->
  <gumpRepositoryId>crayola</gumpRepositoryId>
  <!-- (OPTIONAL) Project description -->
  <description>...</description>
  <!-- (OPTIONAL) Short project description -->
  <shortDescription>...</shortDescription>
  <!-- (OPTIONAL) Project site URL -->
  <url>http://w3.markers.com/crayola</url>
  <!-- (OPTIONAL) Issue-tracking system URL -->

<issueTrackingUrl>http://w3.markers.com/jira/crayola</issueTrackingUrl>
  <!-- (OPTIONAL) Project site address. -->
  <siteAddress>w3.markers.com</siteAddress>
  <!-- (OPTIONAL) Project-site deploy directory (physical location) -->
  <siteDirectory>/www/crayola/site/</siteDirectory>
  <!-- (OPTIONAL) Project distribution directory (physical location) -->

<distributionDirectory>/www/crayola/builds/</distributionDirectory>
  <!-- (OPTIONAL) Project source-repository information -->
```

```
  <repository>

<connection>
scm:cvs:pserver:anoncvs@cvs.markers.com:/home/cvspublic:crayola
</connection>
    <url>http://cvs.markers.com/viewcvs/crayola/</url>
  </repository>
  <!-- (OPTIONAL) Mailing list information -->
  <mailingLists>
    <mailingList>
      <name>Dev List</name>

<subscribe>dev-subscribe@crayola.markers.com</subscribe>

<unsubscribe>dev-unsubscribe@crayola.markers.com</unsubscribe>
    </mailingList>
    ...
  </mailingLists>
  <!-- Developers involved in this project -->
  <developers>
    <developer>
      <name>John Smith</name>
      <id>jsmith</id>
      <email>jsmith@markers.com</email>
    </developer>
    ...
  </developers>
```

将清单 3 中的信息与一个中央构件资源库一起使用，将消除几个常见的构建问题（包括错误配置的 CLASSPATH 或相关性-版本不匹配）。

清单 **3.** 项目相关性部分

```
  <!--
---------------------------------------------------------------- -->
  <!-- Project dependency section                                 -->
  <!--
---------------------------------------------------------------- -->
  <dependencies>
    <!-- This project depends on the JAR file "commons-beanutils-1.5.jar"
          in the Maven repository's commons-beanutils/jars subdirectory
          (more about repository later). -->
```

```
    <dependency>
      <groupId>commons-beanutils</groupId>
      <artifactId>commons-beanutils</artifactId>
      <version>1.5</version>
    </dependency>
    <!-- This project depends on the JAR file "commons-lib-2.1.jar" in
          the Maven repository's markers/jars subdirectory. -->
    <dependency>
      <groupId>markers</groupId>
      <artifactId>commons-lib</artifactId>
      <version>2.1</version>
    </dependency>
  </dependencies>
```

项目构建和报告部分（如清单 4 所示）包含用于配置某些 Maven 插件的重要构建和报告信息。例如，可以配置 Maven 在站点文档生成时包含还是排除某些报告。

清单 **4.** 项目构建部分

```
  ...
  <!--
---------------------------------------------------------------------- -->
  <!-- Project build and reports section                               -->
  <!--
---------------------------------------------------------------------- -->
  <build>
    <!-- (OPTIONAL) Build notification email address. -->
    <nagEmailAddress>jsmith@markers.com</nagEmailAddress>
    <!-- (OPTIONAL) Defines where the Java source resides. -->
    <sourceDirectory>src/java</sourceDirectory>
    <!-- (OPTIONAL) Defines where the Java source for unit test-cases
          resides. -->

<unitTestSourceDirectory>test/java</unitTestSourceDirectory>
    <!-- (OPTIONAL) Unit test-case file pattern. -->
    <unitTest>
      <includes>
        <include>**/*Test.java</include>
      </includes>
    </unitTest>
    <!-- (OPTIONAL) Resources packaged inside the JAR file. -->
    <resources/>
```

```
    <!-- (OPTIONAL) The reports tag lets you select which reports you
          want generated for your site. In this case, only the checkstyle
          report will generate. -->
  </build>
  <reports>
    <report>
      maven-checkstyle-plugin
    </report>
  </reports>
```

项目依靠库来实现其功能。例如，您的项目可能依靠 log4j 进行日志记录，依靠 Xalan 进行 XSLT 转换。对于 J2EE 项目，Web 组件可能依靠 EJB 组件来执行业务操作。Maven 可以让您用它的 POM 来表示不同的相关性。您可以用表 2 所示的标记在 project.xml 文件中描述每一个相关性。

表 **2.** 项目相关性部分

groupId	告诉 Maven 资源库内哪个子目录中包含相关性文件。
artifactId	告诉 Maven 该构件的唯一标识。
version	表示相关性的版本号。
jar	（可选的）表示相关性的 JAR 文件。在绝大多数情况下，可以从相关性的 `<artifactId>` 和 `<version>` 构造 JAR 文件的名称。
type	（可选的）相关性的类型；如 jar 和分发版等。缺省值是 jar。
url	（可选的）相关性项目的 URL，在相关性是在因特网上找到的第三方库时非常有用。

资源库

资源库是另一个主要的 Maven 组件。在有多个项目的基于 Java 的站点中，由第三方库组成的中央资源库常常确保项目之间的一致性。Maven 使资源库的结构符合标准，并且支持驻留在因特网或内部网上的远程资源库。清单 5 显示了资源库的常规结构。

清单 **5.** 资源库

```
repository
|-- ant                   <-- project group ID -->
|   `-- jars              <-- artifact type, followed by 's',
|                         <-- e.g. jars, wars, ears -->
|       `-- ant-1.5.1.jar <-- actual artifact -->
...
```

要创建远程资源库，只需将这个资源库的目录部署在网站中。Maven 建议使用远程资源库以便于集中维护，您将会最大程度地实现项目之间资源的共享。为避免每次构建时都要下载文件，Maven 在首次下载必需的相关性资源时就自动地将其高速缓存在本地资源库中。Maven 将表 3 中所示的特性用于远程资源库和本地资源库。

表 **3.** 用于远程资源库和本地资源库的特性

`maven.repo.remote`	用以逗号分隔的 URL 列表指定远程资源库；缺省情况下使用 `http://www.ibiblio.org/maven`。
`maven.proxy.host`、 `maven.proxy.port`、 `maven.proxy.username` 和 `maven.proxy.password`	如果位于防火墙后面并且需要代理认证才能访问因特网，这些设置将派上用场。
`maven.repo.local`	指定已下载的相关资源的高速缓存位置，缺省值为 `${MAVEN_HOME}/repository` 。在 UNIX 环境中，为了与多个团队共享资源库目录，可以为开发人员创建一个特殊的组，然后给予这个组对资源库目录的读／写访问权。

goal

> **Maven** 中的 **Ant** 任务
>
> Maven 中的 goal 可在其定义中包含任何有效的 Ant 任务，这一点有助于您快速掌握 Maven 以及保护您的 Ant 投入。

Maven 中的 goal 类似 Ant 中的 `target` 。两者都包含实现 goal（或 target）时会执行的任务。要在命令行中实现特定的 goal，可输入 `maven <goal>` 。

要列出所有已定义的 goal，可使用 `maven -g` 。表 4 列出了常用的 goal。

表 **4.** 常用的 **goal**

`java:compile`	编译所有 Java 源代码。
`jar`	创建已编译的源代码的 JAR 文件。
`jar:install`	将已创建的 JAR 文件发布到本地资源库，使得其它项目可访问该 JAR 文件。
`site`	创建项目站点文档。缺省站点文档包含关于项目的有用信息，如包／类相关性、编码风格一致性、源代码交叉引用、单元测试结果或 Javadoc。要生成的报告列表是可定制的。
`site:deploy`	部署生成的站点文档。

Maven 的 goal 是可扩展和可重用的。知道了这一点后，在编写自己的 goal 之前，可先在 Maven 站点上或 `${MAVEN_HOME}/plugins` 中查看 Maven 插件列表。另一个关于免费 Maven 插件的较佳资源是 SourceForge 上的 Maven 插件项目。（以上各项的链接可在 参考资料中获得）。

如果仍不能找到符合您要求的 goal，Maven 给您两种选择：

- 编写 `<preGoal>` 或 `<postGoal>` 来扩展标准 goal
- 编写自己的 goal

无论选择哪种，都要在项目目录中创建名为 maven.xml 的特殊文件。清单 6 显示了框架 maven.xml。

清单 **6.** 框架 **maven.xml**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project xmlns:j="jelly:core">
  ...
  <goal name=...>
    ... build rules, e.g.
```

```
        <mkdir dir="${test.result.dir}"/>
        <echo>Executing JUnit tests</echo>
        ...
      </goal>
      ...
      <preGoal name=...>
        ...
      </preGoal>
      <postGoal name=...>
        ...
      </postGoal>
    </project>
```

熟悉 Ant 的开发人员会发现 Maven 的 goal（同样还有 preGoal 和 postGoal ）可在其定义中包含任何有效的 Ant 任务，这有助于快速学习 Maven 并保护在 Ant 上的投入。为了给 Ant 任务添加动态性，Maven 也使用 Jelly 脚本编制语言。“ 基础 Jelly 编程”用一个样本 maven.xml 文件介绍 Jelly 脚本编制语言。

编写 **<preGoal>** 和 **<postGoal>**

Ant 的 <target> 与 makefile 规则的相似之处在于：定义了规则以后，前提条件和后置条件是固定的。这使得在多个项目间重用构建规则变得更加困难。例如，某个项目中的 compile target 可能依靠 XDoclet 生成源文件，而另一个 compile target 可能不包括任何先决条件。为了克服这种限制，Maven 提供了两个特殊标记： <preGoal> 和 <postGoal> 。从标记的名称可以看出：preGoal 定义在指定的 goal 之前执行的构建规则。另一方面， postGoal 定义实现指定 goal 之后要执行的构建规则。例如，清单 7 中的 preGoal 指示 Maven 在编译源代码之前用 XDoclet 生成源文件。

清单 **7.** 样本 **preGoal** 部分

```
      <preGoal name="java:compile">
        <attainGoal name="xdoclet:ejbdoclet"/>
      </preGoal>
```

Maven 还提供与 Ant 的 <antcall> 标记相似的 <attainGoal> 标记，以便在确有必要直接实现 goal 的情况（如上例）下使用。

编写自己的 **goal**

如果 goal 是特定于项目的，则可在 maven.xml 文件中定义自己的 goal。这些自定义的 goal 会覆盖其它同名的 goal。如果项目包括子项目，子项目也继承这些 goal。

编写插件

为了在项目间共享 goal，可在 Maven 安装插件目录（ ${MAVEN_HOME}/plugins ）中将其打包为插件。典型的 Maven 插件包含 project.xml 和 plugin.jelly 文件。project.xml 文件描述插件的 POM；plugin.jelly 类似 maven.xml 且包含该插件所展示的 goal。插件可以有自己的资源和相关性信息。预先定义的变量 ${plugin.dir} 让用户引用插件目录中的资源。例如，在清单 8 中所示的插件结构中， ${plugin.dir}/dtd/web-app_2_3.dtd 可访问 plugin.jelly 中的 web-app_2_3.dtd 。

清单 **8.** 样本插件结构

```
ejbjar-plugin-1.0
|-- dtd
```

```
|   |-- application_1_3.dtd
|   |-- ejb-jar_2_0.dtd
|   |-- web-app_2_3.dtd
|-- plugin.jelly
`-- project.xml
```

回页首

安装 **Maven**

最近发行的 Maven 1.0-beta-8 基本上是 1.0 的功能完善版。因为 Maven 开发社区每天都在修正错误，如果您遇到任何无法正常工作的问题，则立即从 CVS（Concurrent Version System，并发版本控制系统）获得最新 Maven 版本，然后自行构建（请参阅 参考资料以获得指示信息）。下载了最新的 Maven 源代码之后，可通过调用以下命令来构建 Maven：

```
ant -f build-bootstrap.xml
(set MAVEN_HOME to where you want Maven to reside and
use Ant 1.5.1 to perform the build)
```

如果在防火墙之后操作，请正确设置以下特性： `maven.proxy.host` 、 `maven.proxy.port` 、 `maven.proxy.username` 和 `maven.proxy.password` 。缺省情况下，Maven 资源库驻留在 `${MAVEN_HOME}/repository` 中；通过将 `maven.repo.local` 特性设置为新位置，可以更改 Maven 资源库的位置。

回页首

样本项目文件

请参阅在这个样本 J2EE 项目中使用的 Maven 项目文件。

样本 **J2EE** 项目

掌握了到目前为止所学的知识后，就可以着手使用 Maven 了。本节描述如何用 Maven 设置一个样本 J2EE 项目。

项目目录布局

在进行详细介绍之前，我先说明一下项目的目录布局。尽管不作要求，但事实证明一致的跨项目目录布局非常有用，因为熟悉了一个项目的开发人员可以轻松地浏览其它项目。更重要的是，一致的目录布局可让您编写通用的构建规则。

Maven 的目录布局指南（请参阅 参考资料）适用于绝大多数项目。作为演示，我使用略微不同的布局，如清单 9 所示。

清单 **9.** 样本项目目录布局

```
project
|
|-- LICENSE.txt
|-- project.properties
|-- maven.xml
```

```
|-- project.xml
|-- src
|   `-- java
|       `-- com/....
|   `-- conf
|       `-- Configuration files for staging environments.
|-- test
|   `-- java
|       `-- com/....
|   `-- conf
|       `-- Configuration files for unit testing environments.
`-- xdocs
    `-- index.xml
```

一个 J2EE 项目通常生成 WAR 文件、EJB JAR 文件和 EAR 文件。因为每种文件都包括自己的相关性信息和源文件，所以应将其作为单独项目来构建。通常，通过将子项目存储为主项目的子目录，来构造这一项目／子项目关系。我们的布局如清单 10 所示。

清单 **10. Maven** 中 **J2EE** 项目的高级目录布局

```
j2ee-project
|
|-- project.xml       - Produces the EAR file
|
|-- util-subproject
|   |
|   `-- project.xml   - Produces the Utility JAR file
|
|-- ejb-subproject
|   |
|   `-- project.xml   - Produces the EJB JAR file
|
`-- web-subproject
    |
    `-- project.xml   - Produces the WAR file
```

项目继承

项目继承让 POM 以类似于对象继承的方式从主 POM 继承 — 由于这些项目之间的细微差别（主要是相关性的差别），这一特性在此尤为重要。项目管理部分可在主 project.xml 中集中维护。要使用项目继承，可使用 project.xml 中的 <extend> 标记（请参阅“ 样本项目文件”中的清单 2）。

样本项目的 **goal**

既然已经定义了 POM，就可以编写它们的 goal。因为这些 goal 使用 POM 中定义的特性，所以在继续之前应该首先理解“ 样本项目文件”中的 project.xml 文件。

Utility 子项目

由于 Utility 子项目生成一个包含源目录中类的 JAR 文件 — 由缺省的 jar:jar goal 即可满足要求，因此这里不需要定制的 goal。

因为 Web 子项目和 EJB 子项目都依靠 Utility 子项目，所以，在构建 Web 子项目和 EJB 子项目之前，应该调用 jar:install goal 以将 Utility 子项目 JAR 文件部署到本地资源库。这样，WAR 子项目和 EJB 子项目就可以正确地解析相关性。

Web 子项目

Web 子项目生成一个 WAR 文件，该文件包含源目录的类、 jsp 目录中的 JSP 文件和 conf 目录中的 web.xml 文件。缺省 war:war goal 有更简单的关于项目目录布局的视图。要重用该 goal，可如下定制其行为:

1. 在项目的 project.properties 文件中，将特性 maven.war.src 和 maven.war.webxml 分别设置为 ${maven.build.dir}/webapp 和 ${maven.src.dir}/conf/web.xml 。这告诉 war:war 在哪里查找 Web 来源（JSP 页面、HTML 静态页面和图像等）和 web.xml 文件。
2. 定义一个将所有 JSP 文件复制到 ${maven.build.dir}/webapp 目录中的 preGoal 。以下 maven.xml 可实现这一效果:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project>
  <preGoal name="war:init">
    <copy todir="${maven.build.dir}/webapp">
      <fileset dir="${maven.src.dir}/jsp" include="*.jsp"/>
    </copy>
  </preGoal>
</project>
```

当调用 war:war goal 时，请注意 Utility JAR 文件和 commons-beanutils JAR 文件都被打包到 WAR 文件。通过查看 project.xml 文件的相关性部分中的 war.bundle.jar 特性，Maven 知道要在 WAR 文件中包括哪个文件。

EJB 子项目

给 EJB JAR 文件打包和给 JAR 文件打包相似。如果项目设置与缺省 ejb goal 不匹配，可应用以上“Web 子项目”一节中所描述的技术。在这个特定例子中，将 ejb-jar.xml 从 conf 目录复制到 ${maven.build.dir}/ejb/META-INF 目录，并将 maven.ejb.src 特性设置为 ${maven.build.dir}/ejb 。

要将相关性 JAR 文件添加到 EJB JAR 的清单类路径（manifest classpath）中，可在相关性部分中使用 ejb.manifest.classpath 特性。

主（EAR）项目

在成功编译并部署了子项目（使用 jar:install 、 war:install 和 ejb:install goal）之后，即可创建最终的 EAR 文件。相关性特性 ear.bundle.jar 、 ear.bundle.ejb 和 ear.bundle.war 告诉 ear 插件要在 EAR 文件中包括哪些文件。（对于 Maven 1.0-beta-8，WAR 文件不是受支持的相关性类型，因此 EAR 插件不能正确地给 WAR 文件打包。解决办法：使用 postGoal 手工更新 EAR 文件。）

reactor：自动构建子项目

构建 J2EE 项目需要大量的工作。事实证明，每次项目更改时重复同样的过程耗费时间而且容易出错。为帮助解决这些问题，Maven 的 reactor 功能部件以正确的顺序自动构建子项目，这样就节省了时间且减少了错误。

清单 11 的 maven.xml 演示了定义 reactor 的方法。

清单 **11.** 样本 **reactor** 定义

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project default="all"
         xmlns:m="jelly:maven">
  <goal name="all">
    <m:reactor basedir="${basedir}"
               includes="*/project.xml"
               goals="install"
               banner="Building"
               ignoreFailures="false"/>
  </goal>
</project>
```

该 reactor 首先在 basedir 目录下搜索 project.xml 文件，然后调用 `install` goal。执行的顺序取决于每个项目中的相关性部分。此外，通常可以在主项目的 maven.xml 文件中定义 reactor。因为 goal 在子项目中继承，所以选择 goal 的名称时要当心。

回页首

参与其中

尽管 Maven 是功能丰富的产品，但它仍处于 beta 测试版阶段。因此错误可能在任何地方突然出现。别慌。要找到解决错误的答案，最好的办法是搜索 Maven 的邮件列表归档（请参阅 参考资料）以了解相关的通告。如果没有任何发现，试着将问题公布到邮件列表，以便得到别人的建议。邮件列表上列出的人一般都乐于给予帮助。

要正式地报告错误，请访问 Maven 项目的问题跟踪系统（请参阅 参考资料）。

一旦熟悉了 Maven，您可能会在插件的实现中找到绝大多数答案。当您达到专家水平并且相信 Maven 大有前途的时候，请向社区提供补丁，帮助 Maven 成长。

回页首

结束语

随着如今的项目变得越来越复杂，我们需要能帮助我们表示并管理这些复杂性的工具。Maven 将项目对象模型与功能强大的 XML 脚本编制语言相结合，为我们提供了这样的工具。在本文中，您已经了解如何定义 POM 以及如何使用 Maven 的 `goal` 机制构建项目。我们还研究了使用 Jelly 定制构建行为的不同方法。最后，利用一个样本 J2EE 项目，我们将概念在实际中加以应用。希望您下载

Maven 以推动其发展。

作者感谢 *Jason van Zyl* 对本文的评审。

参考资料

- 您可以参阅本文在 developerWorks 全球站点上的 英文原文.
- 有关 Maven 及其各种插件的更多信息，请访问 Maven网站。您将在那里发现有用的参考资料，包括：
  - 推荐的 目录布局
  - 内置插件文档
  - 邮件列表
  - Maven User Guide，针对 1.0-beta-8 更新
- Sourceforge.net 的 Maven 插件项目提供免费 Maven 插件。
- Ant 网站提供对 Ant 任务的出色引用。在 Maven 中可使用绝大多数 Ant 任务。
- 教程“ Using JavaScript with Ant”向您演示如何用 JavaScript 编写动态 Ant 任务。
- Krysalis Centipede是另一个基于 Ant 的构建系统。
- 在“ Extending Ant to support interactive builds”（ *developerWorks*，2001 年 11 月）中，Anthony Young-Garner 向您演示如何在构建中加入交互式支持，以便为最终用户提供更顺畅、更灵活的体验。
- Malcolm Davis 撰写的“ 利用 Ant 和 JUnit 进行增量开发”（ *developerWorks*，2000 年 11 月）建议用单元测试改进代码。
- GUMP是连续集成相互从属的项目的工具。
- 您将在 *developerWorks*Java 技术专区找到涉及 Java 编程各个方面的数百篇文章。

关于作者

Charles Chan 是 Finetix LLC 的一名顾问。Charles 的兴趣包括分布式系统、高性能计算、国际化和软件设计模式。在业余时间，他为开放源码社区撰稿。可以通过 charlesc@ibiblio.org 与 Charles 联系。

对本文的评价

- ◉ 太差！ (1)
- ○ 需提高 (2)
- ○ 一般；尚可 (3)
- ○ 好文章 (4)
- ○ 真棒！(5)

建议？

反馈意见

回页首