第 15 章

生成项目站点

本章内容

- ❑ 最简单的站点
- ❑ 丰富项目信息
- ❑ 项目报告插件
- ❑ 自定义站点外观
- ❑ 创建自定义页面
- ❑ 国际化
- ❑ 部署站点
- ❑ 小结

Maven 不仅仅是一个自动化构建工具和一个依赖管理工具，它还能够帮助聚合项目信息，促进团队间的交流。POM 可以包含各种项目信息，如项目描述、版本控制系统地址、缺陷跟踪系统地址、许可证信息、开发者信息等。用户可以让 Maven 自动生成一个 Web 站点，以 Web 的形式发布这些信息。此外，Maven 社区提供了大量插件，能让用户生成各种各样的项目审查报告，包括测试覆盖率、静态代码分析、代码变更等。本章详细介绍如何生成 Maven 站点，以及如何配置各种插件生成项目报告。读完本章，应该能够为自己的项目生成漂亮的 Maven 站点，更便捷、更快速地为团队提供项目当前的状态信息。

15.1 最简单的站点

对于 Maven 2 来说，站点生成的逻辑是 Maven 核心的一部分。鉴于灵活性和可扩展性考虑，在 Maven 3 中，这部分逻辑已从核心中移除。由于此设计的变动，Maven 3 用户必须使用 3.x 版本的 maven-site-plugin。例如：

```
<pluginManagement>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-site-plugin</artifactId>
            <version>3.0-beta-1</version>
        </plugin>
    </plugins>
</pluginManagement>
```

Maven 2 用户则应该使用 maven-site-plugin 最新的 2.x 版本。例如：

```
<pluginManagement>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-site-plugin</artifactId>
            <version>2.1.1</version>
        </plugin>
    </plugins>
</pluginManagement>
```

配置了正确版本的 maven-site-plugin 之后，在项目下运行 **mvn site** 就能直接生成一个最简单的站点，用户可以看到如下方代码所示的命令行输出。

```
[INFO] [site:site {execution: default-cli}]
[WARNING] No URL defined for the project-decoration links will not be resolved
[INFO] Generating "Plugin Management" report.
[INFO] Generating "Mailing Lists" report.
[INFO] Generating "Continuous Integration" report.
[INFO] Generating "Dependency Management" report.
[INFO] Generating "Project License" report.
[INFO] Generating "Project Team" report.
[INFO] Generating "Source Repository" report.
[INFO] Generating "About" report.
[INFO] Generating "Issue Tracking" report.
```

```
[INFO] Generating "Project Summary" report.
[INFO] Generating "Dependency Convergence" report.
[INFO] Generating "Dependencies" report.
```

待 Maven 运行完毕后，可以在项目的 target/site/目录下找到 Maven 生成的站点文件，包括 dependencies. html、dependency-convergence. html、index. html 等文件和 css、images 文件夹。读者能够从这些文件及文件夹的名字中猜到其中的内容：css 和 images 文件夹是用来存放站点相关的图片和 css 文件的，其他 html 文件基本对应了一项项目信息，如 dependencies. html 包含了项目依赖信息，license. html 包含了项目许可证信息。index. html 则是站点的主页面，用浏览器打开就能看到图 15-1 所示的页面。

从图 15-1 中可以看到，左边导航栏的下方包含了各类项目信息的链接，包括持续集成、依赖、问题追踪、邮件列表、团队、源码库等。

如果这是一个聚合项目，导航栏的上方还会包含子模块的链接，但是如果单击这些链接，将无法转到子模块的项目页面。这是由于多模块 Maven 项目本身的目录结构导致的。如果将站点发布到服务器上，该问题会自然消失。如果想在本地查看结构正确的站点，则可以 maven-site-plugin 的 stage 目标，将站点预发布至某个本地临时目录下。例如：

```
$ mvn site:stage  -DstagingDirectory=D:\tmp
```

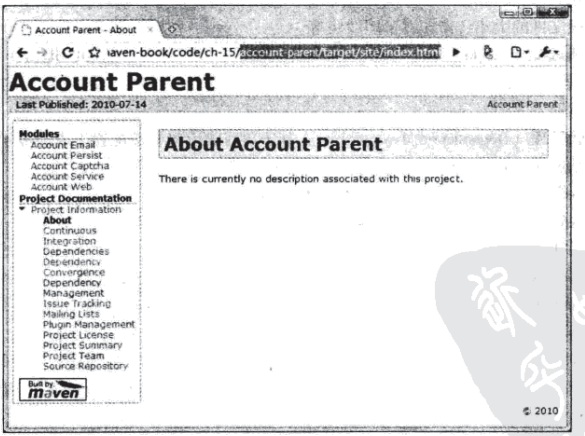


图 15-1　最简单的 Maven 站点

上述命令表示生成项目站点，并预发布至 D:\tmp 目录。读者可以到该目录下找到项目站点的 html 文件，父子模块之间的链接也是可用的。

回顾 7.2.4 节，我们知道 site 生命周期有四个阶段，它们分别为 pre-site、site、post-site

和 site-deploy。其中，pre-site 和 post-site 默认没有绑定任何插件目标，可以说它们是预留给用户做一些站点生成之前及之后的处理的；site 阶段绑定到了 maven-site-plugin 的 site 目标，该目标负责生成项目站点，因此之前使用简单的 mvn site 命令就能直接生成项目站点；site-deploy 目标绑定了 maven-site-plugin 的 deploy 目标，该目标负责将站点部署至远程服务器。本章稍后会详细解释自动化站点部署。

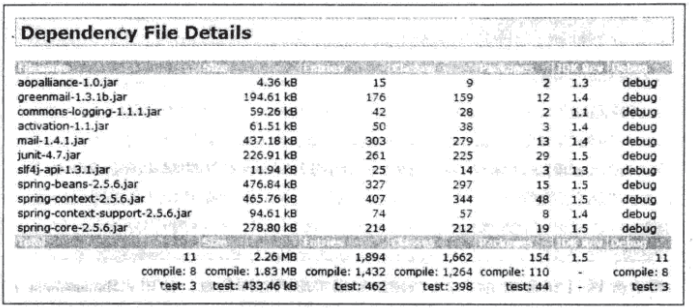
15.2 丰富项目信息

在 15.1 节中可以看到，在默认情况下 Maven 生成的站点包含了很多项目信息链接，这其实是由一个名为 maven-project-info-reports-plugin 的插件生成的。在 Maven 3 中，该插件的配置内置在 maven-site-plugin 中，而在 Maven 2 中，该插件的配置内置在核心源码中。因此你不需要任何配置就能让 Maven 帮你生成项目信息。该插件会基于 POM 配置生成下列项目信息报告：

- **关于**（about）：项目描述。
- **持续集成**（Continuous Integration）：项目持续集成服务器信息。
- **依赖**（Dependencies）：项目依赖信息，包括传递性依赖、依赖图、依赖许可证以及依赖文件的大小、所包含的类数目等。
- **依赖收敛**（Dependency Convergence）：只针对多模块项目生成，提供一些依赖健康状况分析，如各模块使用的依赖版本是否一致、项目中是否有 SNAPSHOT 依赖。
- **依赖管理**（Dependency Management）：基于项目的依赖管理配置生成的报告。
- **问题追踪**（Issue Tracking）：项目的问题追踪系统信息。
- **邮件列表**（Mailing Lists）：项目的邮件列表信息。
- **插件管理**（Plugin Management）：项目所使用插件的列表。
- **项目许可证**（Project License）：项目许可证信息。
- **项目概述**（Project Summary）：项目概述包括坐标、名称、描述等。
- **项目团队**（Project Team）：项目团队信息。
- **源码仓库**（Source Repository）：项目的源码仓库信息。

上述有些项是根据项目已有的依赖和插件配置生成的。例如，依赖这一项就很有意思，除了依赖坐标、传递性依赖以及依赖图，可以使用 maven-dependency-plugin 生成的信息之外，报告还有依赖文件细节的信息，这里详细罗列了每个依赖文件的名称、大小、所包含文件数目、类数目、包数目和 JDK 版本等信息，如图 15-2 所示。

依赖相关的项是基于 POM 的 dependencies 和 dependencyManagement 元素生成的，类似地，其他项也都有其对应的 POM 元素。Maven 不会凭空生成信息，只有用户在 POM 中提供了相关配置后，站点才有可能包含这些信息的报告。为了让站点包含完整的项目信息，需配置 POM，如代码清单 15-1 所示。



Dependency File Details

aopalliance-1.0.jar	4.36 kB	15	9	2	1.3	debug
greenmail-1.3.1b.jar	194.61 kB	176	159	12	1.4	debug
commons-logging-1.1.1.jar	59.26 kB	42	28	2	1.1	debug
activation-1.1.jar	61.51 kB	50	38	3	1.4	debug
mail-1.4.1.jar	437.18 kB	303	279	13	1.4	debug
junit-4.7.jar	226.91 kB	261	225	29	1.5	debug
slf4j-api-1.3.1.jar	11.94 kB	25	14	3	1.3	debug
spring-beans-2.5.6.jar	476.84 kB	327	297	15	1.5	debug
spring-context-2.5.6.jar	465.76 kB	407	344	48	1.5	debug
spring-context-support-2.5.6.jar	94.61 kB	74	57	8	1.4	debug
spring-core-2.5.6.jar	278.80 kB	214	212	19	1.5	debug
11	2.26 MB	1,894	1,662	154	1.5	11
compile: 8	compile: 1.83 MB	compile: 1,432	compile: 1,264	compile: 110	-	compile: 8
test: 3	test: 433.46 kB	test: 462	test: 398	test: 44	-	test: 3

图 15-2　依赖文件细节报告

代码清单 15-1　包含完整项目信息的 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.account < /groupId >
    <artifactId > account-parent < /artifactId >
    <version >1.0.0-SNAPSHOT < /version >
    <packaging > pom < /packaging >
    <name > Account Parent < /name >
    <url > http://mvnbook.juvenxu.com/< /url >
    <description > A project used to illustrate Maven's features. < /description >

    <scm >
        <connection > scm:svn:http://svn.juvenxu.com/mvnboook/trunk < /connection >
        <developerConnection > scm:svn:https://svn.juvenxu.com/mvnboook/trunk
< /developerConnection >
        <url > http://svn.juvenxu.com/mvnboook/trunk < /url >
    < /scm >

    <ciManagement >
        <system > Hudson < /system >
        <url > http://ci.juvenxu.com/mvnbook < /url >
    < /ciManagement >

    <developers >
        <developer >
            <id > juven < /id >
            <name > Juven Xu < /name >
            <email > juvenshun@ gmail.com < /email >
            <timezone > 8 < /timezone >
        < /developer >
    < /developers >
```

```
    <issueManagement>
        <system>JIRA</system>
        <url>http://jira.juvenxu.com/mvnbook</url>
    </issueManagement>

    <licenses>
        <license>
            <name>Apache License, Version 2.0</name>
            <url>http://www.apache.org/licenses/LICENSE-2.0</url>
        </license>
    </licenses>
...
</project>
```

代码清单 15-1 中使用 scm 元素为项目添加了源码仓库信息，使用 ciManagement 元素为项目添加了持续集成服务器信息，使用 developers 元素为项目添加了项目成员团队信息，使用 issueManagement 元素为项目添加了问题追踪系统信息，使用 licenses 元素为项目添加了许可证信息。这时再重新生成站点，相关信息就会体现在站点的项目信息报告中。图 15-3 就显示了一个典型的源码仓库信息报告。

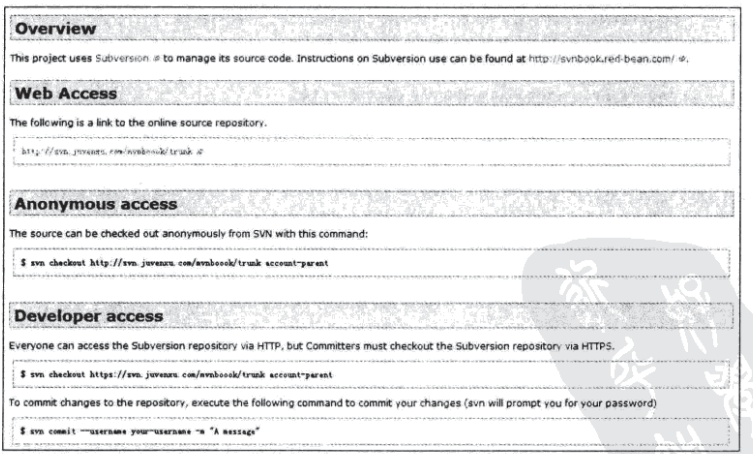


图 15-3 项目源码仓库信息报告

类似的项目信息报告读者可以在很多的开源项目中看到，使用 Maven 站点来一致化开源项目的信息展现方式无疑为用户获取信息提供了便利。

有些时候，用户可能不需要生成某些项目信息项，例如你可能没有邮件列表或者不想在站点中公开源码仓库信息，这时可以配置 maven-project-info-reports-plugin 选择性地生成

信息项，如代码清单 15-2 所示。

代码清单 15-2　选择性地生成项目信息报告

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>2.1.2</version>
        <reportSets>
          <reportSet>
            <reports>
              <report>dependencies</report>
              <report>project-team</report>
              <report>issue-tracking</report>
              <report>license</report>
            </reports>
          </reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
</project>
```

上述代码配置了 maven-project-info-reports-plugin。需要注意的是，项目报告插件需要在 reporting 元素下的 plugins 元素下进行配置，下一节还将介绍其他项目报告插件，也都在这里进行配置。代码清单 15-2 中的配置使得站点的项目信息只包含依赖、团队、问题追踪系统和许可证几项信息，读者可以根据自己的实际情况选择要生成的项目信息。

15.3　项目报告插件

除了默认的项目信息报告，Maven 社区还提供了大量报告插件，只要稍加配置，用户就能让 Maven 自动生成各种内容丰富的报告。下面介绍一些比较常用的报告插件。值得注意的是，报告插件在 POM 中配置的位置与一般的插件配置不同。一般的插件在 <project> <build> <plugins> 下配置，而报告插件在 <project> <reporting> <plugins> 下配置。

15.3.1　JavaDocs

这可能是最简单、也最容易理解的报告插件了。maven-javadoc-plugin 使用 JDK 的 javadoc 工具，基于项目的源代码生成 JavaDocs 文档。该插件的配置如代码清单 15-3 所示。

代码清单 15-3　配置 maven-javadoc-plugin 插件

```
<reporting>
  <plugins>
    <plugin>
```

```
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.7</version>
      </plugin>
      ...
    </plugins>
  </reporting>
```

基于上述简单的配置，当用户使用 mvn site 命令生成站点时，就能得到项目主源码和测试源码的 JavaDocs 文档，如图 15-4 所示。

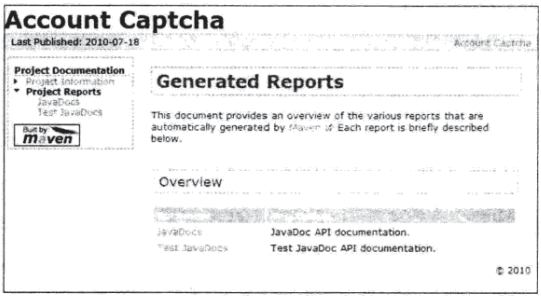


图 15-4 插件报告列表

图 15-4 左侧的导航栏有两个类别，Project Information 包含了 15.2 节讲述的各类基本信息，Project Reports 则包含其他插件生成的报告。这里能看到 maven-javadoc-plugin 生成 JavaDocs 和 Test JavaDocs 文档，单击相应链接就能查看具体文档，如图 15-5 所示。

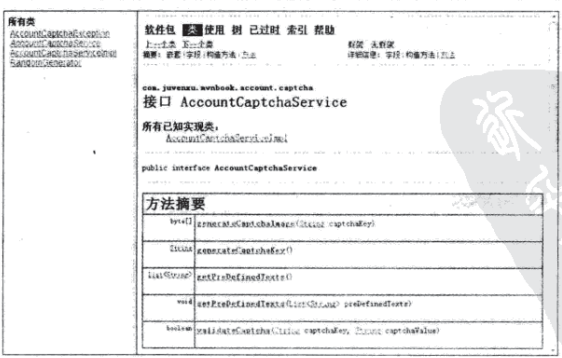


图 15-5 JavaDocs 文档

在生成项目站点文档的时候，一个常见的问题是：用户往往只希望在聚合项目一次性生成融合了所有模块信息的文档，而不是为每个模块单独生成，原因就是为了方便。用户总是希望在一个地方看到尽可能全面的信息，而非不停地单击链接。幸运的是，maven-javadoc-plugin 考虑到了这一点，使用该插件的最新版本，用户无须任何额外的配置，就能在聚合项目的站点中得到包含所有模块的 JavaDocs，配置见代码清单 15-3。

15.3.2　Source Xref

如果能够随时随地打开浏览器访问项目的最新源代码，那无疑会方便团队之间的交流。maven-jxr-plugin 能够帮助我们完成这一目标，在生成站点的时候配置该插件，Maven 就会以 Web 页面的形式将 Java 源代码展现出来。该插件的配置如代码清单 15-4 所示。

代码清单 15-4　配置 maven-jxr-plugin 插件

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jxr-plugin</artifactId>
      <version>2.2</version>
    </plugin>
    ...
  </plugins>
</reporting>
```

若想在聚合模块整合所有的源码，则需添加额外的 aggregate 配置，如代码清单 15-5 所示。

代码清单 15-5　在聚合项目配置 maven-jxr-plugin 插件

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jxr-plugin</artifactId>
      <version>2.2</version>
      <configuration>
        <aggregate>true</aggregate>
      </configuration>
    </plugin>
    ...
  </plugins>
</reporting>
```

生成的源码交叉引用报告如图 15-6 所示。

在这个源码交叉引用文档中，所有源码文件都通过超链接相连，如果之前配置了 JavaDocs 报告，用户还能直接转到源码文件对应的 JavaDoc。

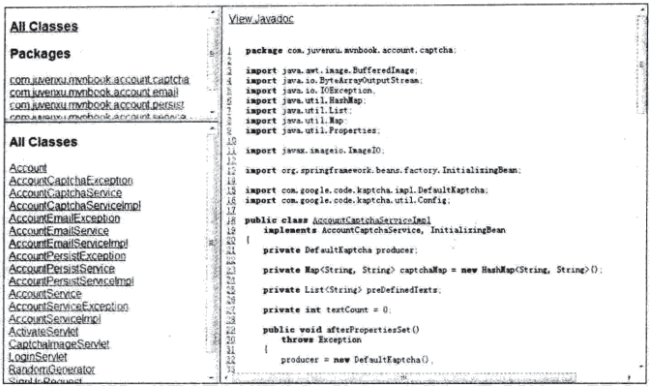


图 15-6 源码交叉引用文档

15.3.3 CheckStyle

CheckStyle 是一个用来帮助 Java 开发人员遵循编码规范的工具，能根据一套规则自动检查 Java 代码，使得团队能够方便地定义自己的编码规范。关于该工具的详细信息可以访问 http://checkstyle.sourceforge.net/进行了解。

要让 Maven 在站点中生成 CheckStyle 报告，只需要配置 maven-checkstyle-plugin，如代码清单 15-6 所示。

代码清单 15-6 配置 maven-checkstyle-plugin 插件

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.5</version>
    </plugin>
    ...
  </plugins>
</reporting>
```

运行 mvn site 命令后，就能得到图 15-7 所示的 CheckStyle 报告。

默认情况下，maven-checkstyle-plugin 会使用 Sun 定义的编码规范，读者能够选择其他预置的规则。也可以自定义规则，maven-checkstyle-plugin 内置了四种规则：

- **config/sun_checks.xml**：Sun 定义的编码规范（默认值）。
- **config/maven_checks.xml**：Maven 社区定义的编码规范。

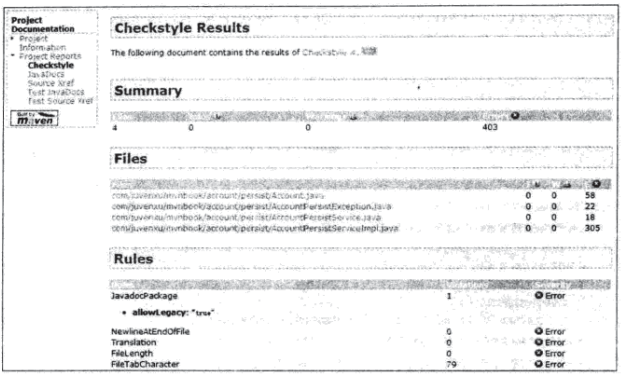


图 15-7　CheckStyle 报告

❑ **config/turbine_checks. xml**：Turbine 定义的编码规范。

❑ **config/avalon_checks. xml**：Avalon 定义的编码规范。

用户可以配置 maven-checkstyle-plugin 使用上述编码规范，如代码清单 15-7 所示。

代码清单 15-7　配置 maven-checkstyle-plugin 使用非默认编码规范

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.5</version>
      <configuration>
        <configLocation>config/maven_checks.xml</configLocation>
      </configuration>
    </plugin>
    ...
  </plugins>
</reporting>
```

通常用户所在的组织会有自己的编码规范，这时就需要创建自己的 checkstyle 规则文件。如在 src/main/resources/目录下定义一个 checkstyle/my_checks. xml 文件，然后配置 <configLocation>checkstyle/my_checks. xml</configLocation> 即可。maven-checkstyle-plugin 实际上是从 ClassPath 载入规则文件，因此对于它来说，无论规则文件是在当前项目中还是在依赖文件中，处理方式都是一样的。

对于多模块项目来说，使用 maven-checkstyle-plugin 会有一些问题。首先，（到本书编写为止）maven-checkstyle-plugin 还不支持报告聚合。也就是说，用户无法在聚合项目的报

告中得到所有模块的 CheckStyle 报告。想要在各个模块中重用自定义的 checkstyle 规则还需要一些额外的配置。具体过程如下：

1）创建一个包含 checkstyle 规则文件的模块：

```
checkstyle/pom.xml
checkstyle/src/main/resources/checkstyle/my-checks.xml
```

2）在聚合模块配置 maven-checkstyle-plugin 依赖该模块：

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.5</version>
      <dependencies>
        <dependency>
          <groupId>com.juvenxu.mvnbook</groupId>
          <artifactId>checkstyle</artifactId>
          <version>1.0</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

3）在聚合模块配置 maven-checkstyle-plugin 使用模块中的 checkstyle 规则：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.5</version>
      <configuration>
        <configLocation>checkstyle/my_checks.xml</configLocation>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

原理就是先创建一个包含自定义规则文件的依赖，然后将该依赖加入到项目的 ClassPath 中，最后从 ClassPath 载入规则文件。

15.3.4 PMD

PMD 是一款强大的 Java 源代码分析工具，它能够寻找代码中的问题，包括潜在的 bug、无用代码、可优化代码、重复代码以及过于复杂的表达式。关于该工具的详细信息可以访问 http://pmd.sourceforge.net/进行了解。

要让 Maven 在站点中生成 PMD 报告，只需要配置 maven-pmd-plugin 如下：

```
<reporting>
  <plugins>
    <plugin>
```

```
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>2.5</version>
    </plugin>
    ...
  </plugins>
</reporting>
```

运行 mvn site 之后，就能得到图 15-8 所示的 PMD 报告。

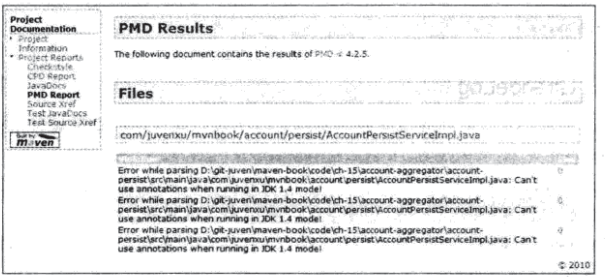


图 15-8　PMD 报告

需要注意的是，除了 PMD 报告之外，maven-pmd-plugin 还会生成一个名为 CPD 的报告，该报告中包含了代码拷贝粘贴的分析结果。

PMD 包含了大量的分析规则，读者可以访问 http://pmd.sourceforge.net/rules/index.html 查看这些规则。PMD 默认使用的规则为 rulesets/basic.xml、rulesets/unusedcode.xml 和 rulesets/importss.xml。要使用其他的规则，可以配置 maven-pmd-plugin 插件，如代码清单 15-8 所示。

代码清单 15-8　配置 maven-pmd-plugin 使用非默认分析规则

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>2.5</version>
      <configuration>
        <rulesets>
          <ruleset>rulesets/braces.xml</ruleset>
          <ruleset>rulesets/naming.xml</ruleset>
          <ruleset>rulesets/strings.xml</ruleset>
        </rulesets>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

maven-pmd-plugin 支持聚合报告，只需要如下配置 aggregate 参数即可：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>2.5</version>
      <configuration>
        <aggregate>true</aggregate>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

15.3.5 ChangeLog

maven-changelog-plugin 能够基于版本控制系统中就近的变更记录生成三份变更报告，它们分别为：

- **Change Log**：基于提交的变更报告，包括每次提交的日期、文件、作者、注释等信息。
- **Developer Activity**：基于作者的变更报告，包括作者列表以及每个作者相关的提交次数和涉及文件数目。
- **File Activity**：基于文件的变更报告，包括变更的文件列表及每个文件的变更次数。

想要生成项目的变更报告，首先需要配置正确的 SCM 信息[⊖]，如下：

```
<project>
...
  <scm>
    <connection>scm:svn:http://192.168.1.103/app/trunk</connection>
    <developerConnection>scm:svn:https://192.168.1.103/app/trunk</develo-
perConnection>
    <url>http://192.168.1.103/account/trunk</url>
  </scm>...
</project>
```

有了 SCM 配置，就可以配置 maven-changelog-plugin 生成变更报告。如下：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-changelog-plugin</artifactId>
      <version>2.2</version>
    </plugin>
  </plugins>
</reporting>
```

生成的变更报告如图 15-9 所示。

默认情况下，maven-changelog-plugin 生成最近 30 天的变更记录，不过用户可以修改该默认值。如下：

⊖ 如果不熟悉该配置，可以回顾 13.4 节。

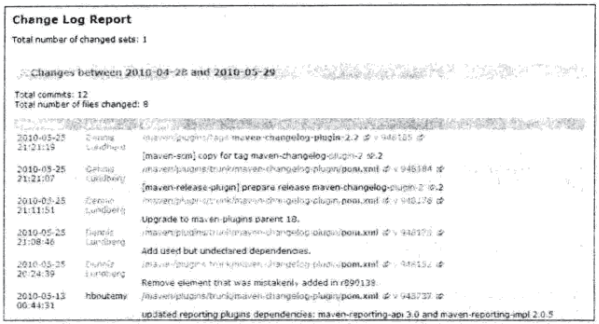


图 15-9　变更报告

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-changelog-plugin</artifactId>
      <version>2.2</version>
      <configuration>
        <type>range</type>
        <range>60</range>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

15.3.6　Cobertura

10.6.2 节已经介绍过用 Cobertura 生成测试覆盖率报告，现在介绍如何将该报告集成到项目站点中。

要在 Maven 站点中包含 Cobertura 测试覆盖率报告，只需要配置 cobertura-maven-plugin，如下：

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>cobertura-maven-plugin</artifactId>
      <version>2.4</version>
    </plugin>
  </plugins>
</reporting>
```

生成的 Cobertura 测试覆盖率报告如图 15-10 所示。

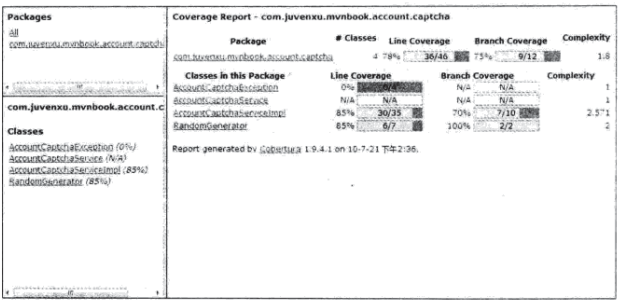


图 15-10 Cobertura 测试覆盖率报告

可以从图 15-10 中看到每个包、每个类的代码行覆盖率和分支覆盖率，单击具体的类还能看到该类代码的具体覆盖情况。可惜的是，到本书编写时为止，cobertura-maven-plugin 还不支持报告聚合，因此用户无法在聚合模块查看所有模块的测试覆盖情况。

15.4 自定义站点外观

Maven 生成的站点非常灵活，除了本章前面提到的标准项目信息报告和其他插件生成的报告，用户还能够自定义站点的布局和外观。这些特性能让用户创建出更适合自己的，更有个性的 Maven 站点。

15.4.1 站点描述符

要自定义站点外观，用户必须创建一个名为 site.xml 的站点描述符文件，且默认该文件应该位于项目的 src/site 目录下。该站点描述符文件是由 XML Schema 约束定义的，相关的 xsd 文件位于 http://maven.apache.org/xsd/decoration-1.0.0.xsd。

一个简单的站点描述符文件如代码清单 15-9 所示。

代码清单 15-9 站点描述符文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/DECORATION/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/DECORATION/1.0.0
    http://maven.apache.org/xsd/decoration-1.0.0.xsd">
  <bannerLeft>
      <name>Account</name>
      <src>images/apache-maven-project.png</src>
      <href>http://maven.apache.org</href>
  </bannerLeft>
```

```
  <body>
    <menu ref="reports"/>
  </body>
  <skin>
    <groupId>com.googlecode.fluido-skin</groupId>
    <artifactId>fluido-skin</artifactId>
    <version>1.3</version>
  </skin>
</project>
```

该描述符文件定义了一个站点头部横幅图片、一个导航栏菜单项以及一个站点皮肤。下面详细介绍各类可在站点描述符中定义的内容。

15.4.2　头部内容及外观

默认情况下，Maven 站点的标题来自于 POM 中的 name 元素值，用户可以配置站点描述符 project 元素的 name 属性来更改此标题。如下：

```
<project name="A Project for Maven Book">
  ...
</project>
```

显示效果如图 15-11 所示。

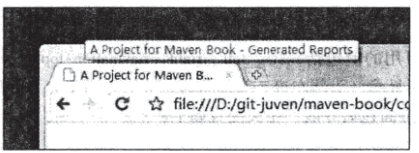


图 15-11　自定义站点标题的效果

如果不进行额外的配置，站点头部左边会显示项目的名称，但是用户可以使用 bannerLeft 元素配置该位置显示自定义的横幅图片。类似地，bannerRight 元素能用来配置显式在头部右边的横幅图片。具体配置如下：

```
<project>
  <bannerLeft>
    <name>maven</name>
    <src>http://maven.apache.org/images/apache-maven-project.png</src>
    <href>http://maven.apache.org</href>
  </bannerLeft>
  <bannerRight>
    <name>java</name>
    <src>images/java.jpg</src>
    <href>http://www.java.com</href>
  </bannerRight>
</project>
```

上述代码为头部配置了两个横幅图片，左边的图片直接引用了 Maven 站点，而右边则使用了本地图片。显示效果如图 15-12 所示。

图 15-12 站点头部横幅图片显示效果

需要注意的是，上述 Java 图片的 src 为 images/java.jpg，是一个本地图片，所有站点使用的本地 Web 资源都必须位于 src/site/resources 目录下。到目前为止，该站点的目录结构是这样的：

```
-src/
  + site/
     + resources/
     |  + images/
     |     + java.jpg
     |
     + site.xml
```

除了标题和头部横幅图片外，Maven 用户还能够配置是否显示站点的最近发布时间和版本。如下：

```
<project>
  <version position="right"/>
  <publishDate position="right"/>
</project>
```

这里的 position 可用的值包括 none、left、right、navigation-top、navigation-bottom 和 bottom，它们分别表示不显示、头部左边、头部右边、导航边栏上方、导航边栏下方和底部。

Maven 站点还支持面包屑导航。相关配置如下：

```
<project>
  <body>
    <breadcrumbs>
      <item name="Maven" href="http://maven.apache.org"/>
      <item name="Juven Xu" href="http://www.juvenxu.com"/>
    </breadcrumbs>
  </body>
</project>
```

显示效果如图 15-13（图中还包括了发布日期和版本）所示。

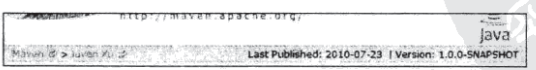


图 15-13 站点的面包屑导航显示效果

15.4.3 皮肤

如果觉得自定义站点标题、横幅图片和面包屑导航等内容还无法满足自己的个性化需求，这时也许可以考虑选择一款非默认的站点皮肤，以将自己的站点与其他站点很明显地区分开来。

自定义站点皮肤分为两步：第一步是选择要使用的站点皮肤构件；第二步是配置站点描述符的 skin 元素使用该构件。

Maven 官方提供了三款皮肤，它们分别为：

- org. apache. maven. skins：maven-classic-skin
- org. apache. maven. skins. maven-default-skin
- org. apache. maven. skins. maven-stylus-skin

其中，maven-default-skin 是站点的默认皮肤，读者可以访问中央仓库以了解这些皮肤的最新版本[㊀]。

除了官方的皮肤，互联网上还有大量的第三方用户创建的站点皮肤。这里笔者要介绍一款托管在 GoogleCode 上的名为 fluido-skin 的皮肤，它非常清爽、简洁，读者可以访问该项目主页[㊁]了解其最新的版本。

下面就以 fluido-skin 为例，配置站点皮肤。编辑 site. xml 如下：

```
<project>
  <skin>
    <groupId>com.googlecode.fluido-skin</groupId>
    <artifactId>fluido-skin</artifactId>
    <version>1.3</version>
  </skin>
</project>
```

图 15-14 显示了使用了 fluido-skin 皮肤后的站点显示效果，看起来与默认的皮肤感觉很不一样。

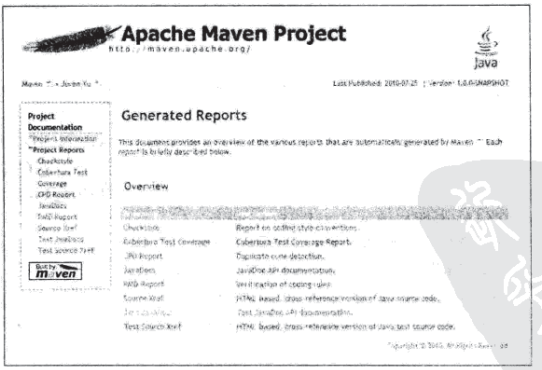


图 15-14　使用了 fluido-skin 皮肤的站点

㊀ 请参考：http://repo1. maven. org/maven2/org/apache/maven/skins/。

㊁ 请参考：http://code. google. com/p/fluido-skin/。

15.4.4 导航边栏

如果用户不自定义站点描述符文件，页面左边的边栏只会显示包含项目信息报告和其他报告的菜单。然而该导航栏内容也是能够自定义的，用户可以在这里创建其他菜单。

要在导航边栏加入自定义菜单，只需要编辑站点描述符中 body 元素下的 menu 子元素。如代码清单 15-10 所示。

代码清单 15-10

```
<project>
  <body>
    <menu name="${project.name}">
      <item name="Introducation" href="introduction.html"/>
      <item name="Usage" href="usage.html"/>
      <item name="FAQ" href="faq.html"/>
    </menu>
    <menu name="Examples">
      <item name="Example 1" href="example_1.html"/>
      <item name="Example 2" href="example_2.html"/>
    </menu>
    <menu ref="reports"/>
  </body>
</project>
```

上述代码中定义了三个菜单，分别为 ${project.name}、Examples 和 reports。

第一个菜单名称使用了 Maven 属性，站点描述符中的 Maven 属性会被自动解析至对应的值。因此这里的 ${project.name} 在站点中会被显示成项目名称，该菜单包含了 3 个子项，分别为 Introduction、Usage 和 FAQ，每个子项链接一个 html 文件（15.5 节将介绍如何创建这些 html 页面）。

第二个菜单名称是 Examples，包含两个子项 Example 1 和 Example 2，也分别链接两个 html 页面。

最后一个菜单比较特殊，它使用的是 ref 属性而非 name 属性，ref 用来引用 Maven 站点默认生成的页面。例如，这里的 reports 表示引用项目报告菜单。除此之外，还有两个可用的 ref 值：parent 表示包含父模块链接的菜单，modules 表示一个包含所有子模块链接的菜单。

基于代码清单 15-10 生成的站点如图 15-15 所示。

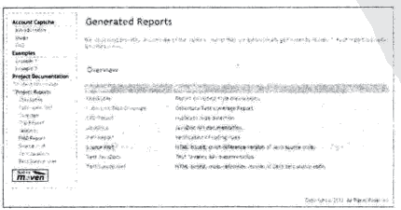


图 15-15 自定义导航边栏菜单

15.5　创建自定义页面

15.4 节介绍了如何自定义站点导航菜单并链接至特定的 html 页面，本节介绍如何创建自定义的站点页面。到目前为止，Maven 支持得比较好的两种文档格式为 APT 和 FML。

APT（Almost Plain Text）是一种类似于维基的文档格式，用户可以用它来快速地创建简单而又结构丰富的文档。例如，创建一个对应于 15.4.4 节提到的 introduction.html 的 APT 文档，首先要记住的是：所有 APT 文档必须位于 src/site/apt/目录。这里创建文件 introduction.apt，内容见代码清单 15-11。

代码清单 15-11　创建 APT 文档

```
 -------------
Introduction
 -------------
Juven Xu
 -------------
2010-07-20
 -------------


What is Maven?

  Apache Maven is a software project management and comprehension tool...

Core Maven Concepts

* Coordinates and Dependency

  descriptions for maven coordinates and dependnecy...

* Repository

  There are many kinds of repositories:

  * Local Repository

  * Central Repository

  * Internal Repository Service

* Plugin and Lifecycle

  descriptions for maven plugin and lifecycle...
```

代码清单 15-11 的第一部分是标题，它们必须缩进，且用多个连字号相隔。在接下来的内容中，“What is Maven?”和“Core Maven Concepts”没有缩进，它们是一级小节。“What is Maven?”下面的内容有缩进，表示一个段落。未缩进的且以星号开头的部分表示二级小节，因此上述代码中有 Coordinate and Dependency、Repository 和 Plugin and Lifecycle 3

个二级小节，它们都包含了一些段落，其中 Repository 下面有包含三个项的列表，它们用缩进的星号表示。

上述代码展示了如何编写一个简单的 APT 文档。笔者没有详细介绍所有 APT 文档格式的语法，如果读者有需要，可以参考 http://maven.apache.org/doxia/references/apt-format.html。

上述 APT 文档展现后的效果如图 15-16 所示。

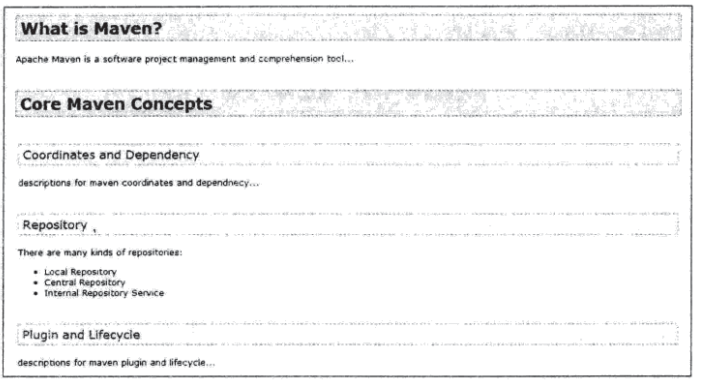


图 15-16 APT 文档效果

FML（FAQ Markup Language）是一种用来创建 FAQ（Frequently Asked Questions，常见问题解答）页面的 XML 文档格式，下面创建一个对应于 15.4.4 节提到的 faq.html 页面的 FML 文档。就像 APT 文档需要放到 src/site/apt/目录一样，FML 文档需要放到 src/site/fml/目录。在这里创建文件 faq.fml，如代码清单 15-12 所示。

代码清单 15-12 创建 FML 文档

```
<?xml version="1.0" encoding="UTF-8"?>
<faqs xmlns="http://maven.apache.org/FML/1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/FML/1.0.1
  http://maven.apache.org/xsd/fml-1.0.1.xsd"
  title="Frequently Asked Questions"
  toplink="false">

  <part id="install">
    <title>Install</title>
    <faq id="download">
      <question>Where to Download?</question>
      <answer>
        <p>Maven: http://maven.apache.org/download.html</p>
```

```
      <p>Nexus: http://nexus.sonatype.org/download-nexus.html</p>
    </answer>
  </faq>
  <faq id="do-install">
    <question>How to Install?</question>
    <answer>
      <p>Description on the installation steps...</p>
    </answer>
  </faq>
</part>

<part id="run">
  <title>Run</title>
  <faq id="how-install">
    <question>How to Run?</question>
    <answer>
      <p>Description on the installation steps...</p>
    </answer>
  </faq>
</part>
</faqs>
```

上述 XML 文档的根元素为 faqs，该元素的 title 属性定义了文档的标题。根元素下面使用 part 元素定义了两个文档部分，第一个是 install，第二个是 run。每个文档部分有自己的标题，以及用 faq 元素定义的问题项目，faq 的子元素 question 用来定义问题，子元素 answer 用来定义答案，这种结构是非常清晰的。同样地，这里不会详细解释所有的 FML 文档语法，如果有需要，可以访问 http://maven.apache.org/doxia/references/fml-format.html。

上述 FML 文档展现后的效果如图 15-17 所示。

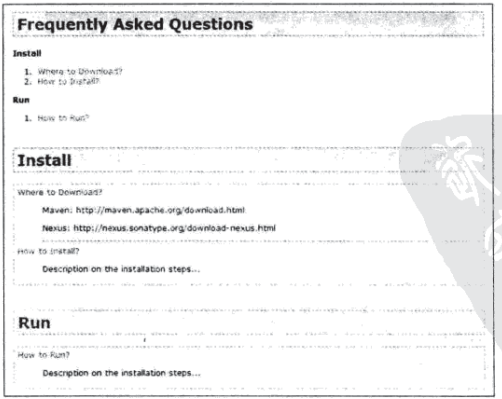


图 15-17　FML 文档效果

到目前为止，站点的目录结构如下：

```
-src/
   + site/
      + resources/
      |  + images/
      |     + java.jpg
      |
      + apt/
      |  + introduction.apt
      |
      + fml/
      |  + fql.fml
      |
      + site.xml
```

15.6 国际化

对于广大欧美以外的用户来说，站点上难免需要添加一些本土的文字，如果没有特殊的配置，站点可能无法对其使用正确的字符集编码。本节以简体中文为例，介绍如何生成本地化的 Maven 站点。

要生成正确的简体中文站点，用户首先需要确保项目所有的源码，包括 pom. xml、site. xml 以及 apt 文档等，都以 UTF-8 编码保存，各种编辑器和 IDE 都支持用户指定保存文档的编码。图 15-18 就展示了 Windows 上用记事本保存文档时候如何指定 UTF-8 编码。

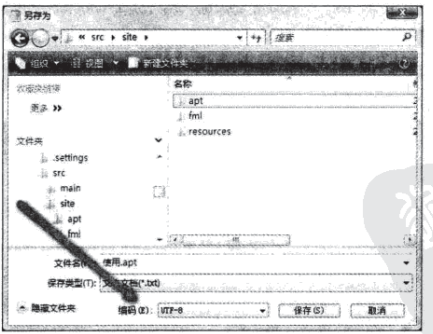


图 15-18 用记事本保存文档时指定 UTF-8 编码

接下来要做的是告诉 maven-site-plugin 使用 UTF-8 编码读取所有源码及文档，并且同样使用 UTF-8 编码呈现站点 html 文档。这两点可以通过配置两个 Maven 属性实现，如下：

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  </properties>
```

project. build. sourceEncoding 属性用来指定 Maven 用什么编码来读取源码及文档，而 project. reporting. outputEncoding 用来指定 Maven 用什么编码来呈现站点的 html 文档。

最后一步要做的是配置 maven-site-plugin 指定当地的语言，配置当地语言为简体中文 zh_CN。如下：

```
<plugins>
...
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-site-plugin</artifactId>
    <version>2.1.1</version>
    <configuration>
      <locales>zh_CN</locales>
    </configuration>
  </plugin>
..
<plugins>
```

完成这些配置后，就能生成图 15-19 所示的中文站点。

图 15-19　生成中文 Maven 站点

15.7　部署站点

为了方便团队和用户得到必要的项目信息，我们需要将 Maven 站点部署到服务器上。Maven 支持多种协议部署站点，包括 FTP、SCP 和 DAV。

如下代码就配置了一个基于 DAV 协议的站点部署地址：

```
<project>
  ...
  <distributionManagement>
    <site>
      <id>app-site</id>
      <url>dav:https://www.juvenxu.com/sites/app</url>
    </site>
  </distributionManagement>
  ...
</project>
```

上述代码中，url 的值以 dav 开头，表示服务器必须支持 WEBDAV。此外，为了确保安全性，服务器的访问一般都需要认证。这个时候就需要配置 settings.xml 文件的 server 元素，这一点与部署构件至 Maven 仓库类似。需要注意的是：要确保 server 的 id 值与 site 的 id 值完全一致。

```
<settings>
  ...
  <servers>
    <server>
      <id>app-site</id>
      <username>juven</username>
      <password>******</password>
    </server>
    ...
  </servers>
  ...
</settings>
```

需要提醒的是，如果在部署的时候遇到问题，请尝试配置最新的 maven-site-plugin。到本书编写时为止，2.x 的最新版本为 2.1.1，3.x 的最新版本为 3.0-beta-2。

如果想要使用 FTP 协议部署站点，那么除了配置正确的部署地址和认证信息外，还需要配置额外的扩展组件 wagon-ftp，如代码清单 15-13 所示。

代码清单 15-13　使用 FTP 协议部署站点

```
<project>
  ...
  <build>
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
        <version>2.1.1</version>
      </plugin>
      ...
    </plugins>
    <extensions>
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
        <artifactId>wagon-ftp</artifactId>
```

```
      <version>1.0-beta-6</version>
    </extension>
  </extensions>
</build>

<distributionManagement>
  <site>
    <id>app-site</id>
    <url>ftp://www.juvenxu.com/site/app</url>
  </site>
</distributionManagement>
...
</project>
```

上述代码中最重要的部分是通过 extension 元素配置了扩展组件 wagon-ftp，有了该组件，Maven 才能正确识别 FTP 协议。该代码中为 maven-site-plugin 和 wagon-ftp 都配置了最新的版本，这么做是为了避免之前版本中存在的一些 bug。

如果希望通过 SCP 协议部署站点，只需要相应地配置 distributionManagement 元素即可。如下：

```
<project>
  ...
  <distributionManagement>
    ...
    <site>
      <id>app-site</id>
      <url>scp://shell.juvenxu.com/home/juven/maven/site/</url>
    </site>
  </distributionManagement>
  ...
</project>
```

与 DAV 和 FTP 不同的是，SCP 协议通常使用密钥进行认证，因此在 settings.xml 中配置认证信息的时候，就可能需要 passphrase 和 privateKey 元素。如下：

```
<settings>
  ...
  <servers>
    <server>
      <id>app-site</id>
      <passphrase>somepassphrase</passphrase>
      <privateKey>C:/sshkeys/id_rsa</privateKey>
    </server>
    ...
  </servers>
  ...
</settings>
```

上述代码中，privateKey 表示私钥的地址，passphrase 表示私钥的口令。

站点部署地址及认证信息配置完成后，只需要输入以下命令就能让 Maven 部署站点了：

```
$ mvn clean site-deploy
```

site-deploy 是 site 生命周期的一个阶段，其对应绑定了 maven-site-plugin 的 deploy 目标，

该目标的工作就是部署 Maven 站点。

15.8 小结

本章详细讲述了如何使用 Maven 生成项目站点，首先介绍了如何快速生成一个最简单的站点，然后在此基础上通过丰富项目信息来丰富站点的内容。用户还能够使用大量现成的插件来生成各种站点报告，包括 JavaDocs、源码交叉引用、CheckStyle、PMD、ChangeLog 以及测试覆盖率报告等。

此外，Maven 还允许用户自定义站点各个部分的外观，甚至更换皮肤。如果用户有自定义的内容想放入站点，则可以编写 APT 或者 FML 文档。

本章还介绍了如何配置 POM 来支持中文的站点。最后，用户可以使用 WEBDAV、FTP 或者 SCP 协议将站点发布到服务器。