



放弃我。抓紧我
STAY WITH ME

王凯陈乔恩演绎时尚圈爱情童话
搜狐视频 正在热播



simonyee's blog 首页 日志

simonyee's blog



博文广场

个人资料

日志正文



Maven最佳实践：划分模块

2013-11-03 23:47

阅读(1404) 评论(0)

“分天下为三十六郡，郡置守，尉，监”——《史记·秦始皇本纪》

所有用Maven管理的真实的项目都应该是分模块的，每个模块都对应着一个pom.xml。它们之间通过继承和聚合（也称作多模块，multi- module）相互关联。那么，为什么要这么做呢？我们明明在开发一个项目，划分模块后，导入Eclipse变成了N个项目，这会带来复杂度，给开发带来不便。

博客年龄：10年10个月
访问：27392次
文章：201篇

个人描述
姓名：***
职业：**
年龄：**
位置：中国，**
个性介绍：

博主最新文章

为了解释原因，假设有这样一个项目，很常见的Java Web应用。在这个应用中，我们分了几层：

- Dao层负责数据库交互，封装了Hibernate交互的类。
- Service层处理业务逻辑，放一些Service接口和实现相关的Bean。
- Web层负责与客户端交互，主要有一些Struts的Action类。

对应的，在一个项目中，我们会看到一些包名：

- org.myorg.app.dao
- org.myorg.app.service
- org.myorg.app.web
- org.myorg.app.util

这样整个项目的框架就清晰了，但随着项目的进行，你可能会遇到如下问题：

[Maven最佳实践：版本管理](#)

[Maven最佳实践：划分模块](#)

[CSS盒子模型之firefox与IE区别篇](#)

[如何解决Java WEB应用中的乱码问题](#)

[Maven2:OutOfMemory when execute mvn command](#)

[更多文章>>](#)

1. 这个应用可能需要有一个前台和一个后台管理端（web或者swing），你发现大部分dao，一些service，和大部分util是在两个应用中可。这样的问题，你一周内遇到了好几次。
2. pom.xml中的依赖列表越来越长以重用的，但是，由于目前只有一个项目（WAR），你不得不新建一个项目依赖这个WAR，这变得非常的恶心，因为在Maven中配置对WAR的依赖远不如依赖JAR那样简单明了，而且你根本不需要org.myorg.app.web。有人修改了dao，提交到 svn并且不小心导致build失败了，你在编写service的代码，发现编译不过，只能等那人把dao修复了，你才能继续进行，很多人都在修改，到后来你根本就不清楚哪个依赖是谁需要的，渐渐的，很多不必要的依赖被引入。甚至出现了一个依赖有多个版本存在。
3. build整个项目的时间越来越长，尽管你只是一直在web层工作，但你不得不build整个项目。
4. 某个模块，比如util，你只想让一些经验丰富的人来维护，可是，现在这种情况，每个开发者都能修改，这导致关键模块的代码质量不能达到你的要求。

我们会发现，其实这里实际上没有遵守一个设计模式原则：“高内聚，低耦合”。虽然我们通过包名划分了层次，并且你还会说，这些包的依赖都是单向的，没有包的环依赖。这很好，但还不够，因为就构建层次来说，所有东西都被耦合在一起了。因此我们需要使用Maven划分模块。

一个简单的Maven模块结构是这样的：

```
---- app-parent
|-- pom.xml (pom)
|
|-- app-util
|   |-- pom.xml (jar)
|
|-- app-dao
|   |-- pom.xml (jar)
|
|-- app-service
|   |-- pom.xml (jar)
|
|-- app-web
|   |-- pom.xml (war)
```

上述简单示意图中，有一个父项目(**app-parent**)聚合很多子项目（**app-util, app-dao, app-service, app-web**）。每个项目，不管是父子，都含有一个**pom.xml**文件。而且要注意的是，小括号中标出了每个项目的打包类型。父项目是**pom**,也只能是 **pom**。子项目有**jar**，或者**war**。根据它包含的内容具体考虑。

这些模块的依赖关系如下：

```
app-dao    --> app-util
app-service --> app-dao
app-web    --> app-service
```

注意依赖的传递性（大部分情况是传递的，除非你配置了特殊的依赖**scope**），**app-dao**依赖于**app-util**，**app-service**依赖于**app-dao**，于是**app-service**也依赖于**app-util**。同理，**app-web**依赖于**app-dao,app-util**。

用项目层次的划分替代包层次的划分能给我们带来如下好处：

1. 方便重用，如果你有一个新的**swing**项目需要用到**app-dao**和**app-service**，添加对它们的依赖即可，你不再需要去依赖一个**WAR**。而有些模块，如**app-util**，完全可以渐渐进化成公司的一份基础工具类库，供所有项目使用。这是模块化最重要的一个目的。
2. 由于你现在划分了模块，每个模块的配置都在各自的**pom.xml**里，不用再到一个混乱的纷繁复杂的总的**POM**中寻找自己的配置。
3. 如果你只是在**app-dao**上工作，你不再需要**build**整个项目，只要在**app-dao**目录运行**mvn**命令进行**build**即可，这样可以节省时间，尤其是当项目越来越复杂，**build**越来越耗时后。
4. 某些模块，如**app-util**被所有人依赖，但你不想给所有人修改，现在你完全可以从这个项目结构出来，做成另外一个项目，**svn**只给特定的人访问，但仍提供**jar**给别人使用。
5. 多模块的**Maven**项目结构支持一些**Maven**的更有趣的特性（如**DepencyManagement**），这留作以后讨论。

接下来讨论一下**POM**配置细节，实际上非常简单，先看**app-parent**的**pom.xml**：

Xml代码

1. `<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
2. `xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">`

```
3.    <modelVersion>4.0.0</modelVersion>
4.    <groupId>org.myorg.myapp</groupId>
5.    <artifactId>app-parent</artifactId>
6.    <packaging>pom</packaging>
7.    <version>1.0-SNAPSHOT</version>
8.    <modules>
9.        <module>app-util</module>
10.       <module>app-dao</module>
11.       <module>app-service</module>
12.       <module>app-web</module>
13.    </modules>
14. </project>
```

Maven的坐标GAV（groupId, artifactId, version）在这里进行配置，这些都是必须的。特殊的地方在于，这里的packaging为pom。所有带有子模块的项目的packaging都为 pom。packaging如果不进行配置，它的默认值是jar，代表Maven会将项目打成一个jar包。

该配置重要的地方在于modules，例子中包含的子模块有app-util, app-dao, app-service, app-war。在Maven build app-parent的时候，它会根据子模块的相互依赖关系整理一个build顺序，然后依次build。

这就是一个父模块大概需要的配置，接下来看一下子模块符合配置继承父模块。、

Xml代码

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
   2001/XMLSchema-instance"
2.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
   maven-v4_0_0.xsd">
3.    <parent>
4.        <artifactId>app-parent</artifactId>
5.        <groupId>org.myorg.myapp</groupId>
6.        <version>1.0-SNAPSHOT</version>
7.    </parent>
8.    <modelVersion>4.0.0</modelVersion>
9.    <artifactId>app-util</artifactId>
10.    <dependencies>
11.        <dependency>
12.            <groupId>commons-lang</groupId>
13.            <artifactId>commons-lang</artifactId>
14.            <version>2.4</version>
15.        </dependency>
16.    </dependencies>
```

17. </project>

app-util模块继承了app-parent父模块，因此这个POM的一开始就声明了对app-parent的引用，该引用是通过Maven坐标GAV实现的。而关于项目app-util本身，它却没有声明完整GAV，这里我们只看到了artifactId。这个POM并没有错，groupId 和version默认从父模块继承了。实际上子模块从父模块继承一切东西，包括依赖，插件配置等等。

此外app-util配置了一个对于commons-lang的简单依赖，这是最简单的依赖配置形式。大部分情况，也是通过GAV引用的。

再看一下app-dao，它也是继承于app-parent，同时依赖于app-util：

Xml代码

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
   2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
   maven-v4_0_0.xsd">
3.   <parent>
4.     <artifactId>app-parent</artifactId>
5.     <groupId>org.myorg.myapp</groupId>
6.     <version>1.0-SNAPSHOT</version>
7.   </parent>
8.   <modelVersion>4.0.0</modelVersion>
9.   <artifactId>app-dao</artifactId>
10.  <dependencies>
11.    <dependency>
12.      <groupId>org.myorg.myapp</groupId>
13.      <artifactId>app-util</artifactId>
14.      <version>${project.version}</version>
15.    </dependency>
16.  </dependencies>
17. </project>
```

该配置和app-util的配置几乎没什么差别，不同的地方在于，依赖变化了，app-dao依赖于app-util。这里要注意的是 version的值为\${project.version}，这个值是一个属性引用，指向了POM的project/version的值，也就是这个 POM对应的version。由于app-dao的version继承于app-parent，因此它的值就是1.0-SNAPSHOT。而app-util也继承了这个值，因此在所有这些项目中，我们做到了保持版本一致。

这里还需要注意的是，app-dao依赖于app-util，而app-util又依赖于commons-lang，根据传递性，app-dao也拥有了对于commons-lang的依赖。

app-service我们跳过不谈，它依赖于app-dao。我们最后看一下app-web：

Xml代码

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
   2001/XMLSchema-instance"
2.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
   maven-v4_0_0.xsd">
3.   <parent>
4.     <artifactId>app-parent</artifactId>
5.     <groupId>org.myorg.myapp</groupId>
6.     <version>1.0-SNAPSHOT</version>
7.   </parent>
8.   <modelVersion>4.0.0</modelVersion>
9.   <artifactId>app-web</artifactId>
10.  <packaging>war</packaging>
11.  <dependencies>
12.    <dependency>
13.      <groupId>org.myorg.myapp</groupId>
14.      <artifactId>app-service</artifactId>
15.      <version>${project.version}</version>
16.    </dependency>
17.  </dependencies>
18. </project>
```

app-web依赖于app-service，因此配置了对其的依赖。

由于app-web是我们最终要部署的应用，因此它的packaging是war。为此，你需要有一个目录src/main/webapp。并在这个目录下拥有web应用需要的文件，如/WEB-INF/web.xml。没有web.xml，Maven会报告build失败，此外你可能还会有这样一些子目录： /js, /img, /css ... 。

看看Maven是如何build整个项目的，我们在 app-parent 根目录中运行 mvn clean install ，输出的末尾会有大致这样的内容：

```
...
...
[INFO] [war:war]
[INFO] Packaging webapp
[INFO] Assembling webapp[app-web] in [/home/juven/workspaces/ws-others/myapp/app-we
b/target/app-web-1.0-SNAPSHOT]
```

```
[INFO] Processing war project
[INFO] Webapp assembled in[50 msec]
[INFO] Building war: /home/juven/workspaces/ws-others/myapp/app-web/target/app-web-1.0-SNAPSHOT.war
[INFO] [install:install]
[INFO] Installing /home/juven/workspaces/ws-others/myapp/app-web/target/app-web-1.0-SNAPSHOT.war to /home/juven/.m2/repository/org/myorg/myapp/app-web/1.0-SNAPSHOT/app-web-1.0-SNAPSHOT.war
[INFO]
[INFO]
[INFO] -----
[INFO] Reactor Summary:
[INFO] -----
[INFO] app-parent ..... SUCCESS [1.191s]
[INFO] app-util ..... SUCCESS [1.274s]
[INFO] app-dao ..... SUCCESS [0.583s]
[INFO] app-service ..... SUCCESS [0.593s]
[INFO] app-web ..... SUCCESS [0.976s]
[INFO] -----
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 4 seconds
[INFO] Finished at: Sat Dec 27 08:20:18 PST 2008
[INFO] Final Memory: 3M/17M
[INFO] -----
```

注意Reactor Summary，整个项目根据我们希望的顺序进行build。Maven根据我们的依赖配置，智能的安排了顺序，app-util, app-dao, app-service, app-web。

最后，你可以在 app-web/target 目录下找到文件 app-web-1.0-SNAPSHOT.war ，打开这个war包，在 /WEB-INF/lib 目录看到了 commons-lang-2.4.jar，以及对应的app-util, app-dao, app-service 的jar包。Maven自动帮你处理了打包的事情，并且根据你的依赖配置帮你引入了相应的jar文件。

使用多模块的Maven配置，可以帮助项目划分模块，鼓励重用，防止POM变得过于庞大，方便某个模块的构建，而不用每次都构建整个项目，并且使得针对某个模块的特殊控制更为方便。本文同时给出了一个实际的配置样例，展示了如何使用Maven配置多模块项目。

分享到：

阅读(1404) 评论(0)

上一篇：[Maven最佳实践：版本管理](#)

下一篇：[CSS盒子模型之firefox与IE区别篇](#)

评论  想第一时间抢沙发么？

由于最近广告泛滥，暂只允许登录用户对此文评论。[登录](#)

[帮助](#) - [客服中心](#) - [意见建议](#) - [举报](#) - [搜狐博客](#) - [搜狐首页](#) - [全部博文](#) - [新闻](#)

Copyright © 2014 Sohu.com Inc. All rights reserved. 搜狐公司 版权所有