

Maven最佳实践 划分模块 配置多模块项目 pom modules

转载 2014年04月30日 16:17:08

标签：maven (http://so.csdn.net/so/search/s.do?q=maven&t=blog) /

划分模块 (http://so.csdn.net/so/search/s.do?q=划分模块&t=blog) /

pom (http://so.csdn.net/so/search/s.do?q=pom&t=blog) /

modules (http://so.csdn.net/so/search/s.do?q=modules&t=blog) /

配置多模块项目 (http://so.csdn.net/so/search/s.do?q=配置多模块项目&t=blog)

6708

原文地址：http://juvenshun.iteye.com/blog/305865
http://blog.csdn.net/woxueliuyun/article/details/9170369
“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。

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Structs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：

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。



0



- 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。



内容举报



返回顶部



0



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>
```



内容举报



返回顶部



```
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>
```



内容举报



返回顶部



0



```
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-web/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] -----
```



内容举报



返回顶部



[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。

Super POM (project object model)

Maven内置了一个默认的POM(不在项目中，因此不可见)，每一个project都会继承自这个默认的POM，因此叫Super POM。除非在项目的配置(POM)中显式的修改，否则使用默认的配置。不同的Maven版本，默认的配置也不一样，遇到问题则需要自己检查。

最小的POM

一个project就是一个artifact，project的全称为: <groupId>:<artifactId>:<version>。
在Maven1中是project.xml，Maven2改成了pom.xml。在Maven1中还有一个maven.xml用于包含可以执行的目标，在Maven2已经配置到了pom.xml中。

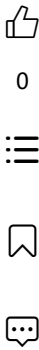
```
[html]
1. <projectxmlnsprojectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchema
   instance"
2.     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
   4.0.0.xsd">
3.     <modelVersion>4.0.0</modelVersion>
4.     <groupId>com.ebay.raptor.samples</groupId>
5.     <artifactId>SamplesParent</artifactId>
6.     <version>1.1.0-SNAPSHOT</version>
7. </project>
```



内容举报



返回顶部



artifact可以是任何东西，包括JAR, WAR,POM, EBA文件等。group只是artifact的命名空间，有点类似于java的包。如果项目还处在开发阶段，在版本后会会有一个"SNAPSHOT"，Maven只允许snapshot artifact被更新，release版本是不能更新的。
modelVersion很重要，因为不同的model，POM的格式是不一样的。
以上几个节点构成了一个最小的POM，这些是POM必须拥有的信息。

继承

如果我们需要把一个artifact放到另一个artifact中，就需要设置继承关系，这个继承关系是由子的module来维护的，因此会在module的pom中有一个parent节点。

```
[html]
1. <project>
2.   <parent>
3.     <groupId>com.mycompany.app</groupId>
4.     <artifactId>my-app</artifactId>
5.     <version>1</version>
6.   </parent>
7.   <modelVersion>4.0.0</modelVersion>
8.   <groupId>com.mycompany.app</groupId>
9.   <artifactId>my-module</artifactId>
10.  <version>1</version>
11. </project>
```

刚才上面讲的是目录结构是如下的时候：

```
| -- my-module
|   `-- pom.xml
`-- pom.xml
```

如果目录结构是下面的样子，

```
| -- my-module
|   `-- pom.xml
|-- parent
|   `-- pom.xml
```

则需要在parent中添加relativePath节点：

```
[html]
1. <project>
2.   <parent>
3.     <groupId>com.mycompany.app</groupId>
4.     <artifactId>my-app</artifactId>
5.     <version>1</version>
6.     <relativePath>../parent/pom.xml</relativePath>
7.   </parent>
8.   <modelVersion>4.0.0</modelVersion>
9.   <artifactId>my-module</artifactId>
10. </project>
```



内容举报



返回顶部

集成(aggregation)

和继承有点类似，但是父子关系是由父的POM来维护的。方法是在POM中添加modules节点：

```
[html]
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.   <groupId>com.mycompany.app</groupId>
4.   <artifactId>my-app</artifactId>
5.   <version>1</version>
6.   <packaging>pom</packaging>
7.   <modules>
8.     <module>my-module</module>
9.   </modules>
10. </project>
```

节点module其实是目录名称，因此如果module不在app中，而是与app同级，则可以写成这样:

```
[html]
1. <modules>
2.   <module>../my-module</module>
3. </modules>
```

项目插值与变量

整个POM相当于一个对象，子节点就是一个个属性，因此可以直接访问:

```
[html]
1. <version>${project.version}</version>
```

另外，还有几个特殊的变量:

project.basedir	当前项目所在的目录
project.baseUri	当前项目所在的目录，只不过用 URI的格式表示. 从Maven 2.1.0开始
maven.build.timestamp	Build的开始时间.从Maven 2.1.0-M1开始

在project也可以自定义自己的变量，方法是在properties中添加属性:

```
[html]
1. <properties>
2.   <mavenVersion>2.1</mavenVersion>
3. </properties>
```



0



内容举报



返回顶部



相关文章推荐

Maven实战（三）多模块项目的POM重构 (/wangmuming/article/details/46915353)

重复，还是重复 程序员应该有狗一般的嗅觉，要能嗅到重复这一最常见的坏味道，不管重复披着怎样的外衣，一旦发现，都应该毫不留情地彻底地将其干掉。不要因为POM不是产品代码而纵容重复在这里发酵，例如这...



wangmuming (<http://blog.csdn.net/wangmuming>) 2015-07-16 17:49 2451

maven 中 pom.xml 配置文件标签说明，dependencyManagement和dependencies区别 (/jiangyu1013/article/details/52424726)

maven 中 pom.xml 配置文件标签说明，dependencyManagement和dependencies区别



u011314442 (<http://blog.csdn.net/u011314442>) 2016-09-03 19:06 2613



太任性！学AI的应届学弟怒拒20K Offer，他想要多少钱？

AI改变命运呀！！前段时间在我司联合举办的秋季招聘会上，一名刚刚毕业的学弟陆续拒绝2份Offer，企业给出18K、23K高薪，学弟拒绝后直接来了一句...

(http://www.baidu.com/cb.php?c=lgF_pyfqnHmsrH61rH60lZ0qnfK9ujYzP1nsrjD10Aw-5Hc3rHnYnHb0TAq15HfLPWRznjb0T1YvP1N-P19BnWDzm16Ln1KB0AwY5HDdnjb1PjT4rjb0lgF_5y9YIZ0lQzq-uZR8mLPbUB48ugfEIAqspynEmybk5LNYUNq1ULNzmvRqmhkEu1Ds0ZFb5HD0mhYqn0KsTWYs0ZNGujYkPHTYn1mk0AqGujYknWb3rjDY0APGujYLnWm4n1c0ULI85H00TZbqnW0v0APzm)

Maven多模块开发之Web工程调试与部署 (/lxf9601/article/details/5765126)

最近学习在eclipse3.6环境下用maven2构建一个多模块的项目分core/dal/dao/web四块,当运行tomcat web工程调试需要把core/dal/dao的工程先mvn insta...



lxf9601 (<http://blog.csdn.net/lxf9601>) 2010-07-25 22:22 10388

Maven多模块项目 eclipse热部署 Maven项目实现 tomcat热部署 (/laoshuisheng/article/details/6420003)

Maven 多模块项目在eclipse下面热部署，即你可以体验下无论你修改整个项目里面的任何模块的代码，都不需要用maven打包就可以看到效果， 1、首先准备好创建一个maven多项目的代码，准备好...



0



内容举报



返回顶部



0



laoshuisheng (<http://blog.csdn.net/laoshuisheng>) 2011-05-14 17:06 19787

maven进阶：一个多模块项目 (/eclipser1987/article/details/5739177)

一个多模块项目通过一个父POM 引用一个或多个子模块来定义。父项目，通过以下配置，将子项目关联。pomsimple-weathersimple-webapp 其中值得注意的是pom这个父项目不...



eclipser1987 (<http://blog.csdn.net/eclipser1987>) 2010-07-16 11:17 39476

基于maven使用IDEA创建多模块项目 (/williamhappy/article/details/54376855)

一 项目工程目录 二 创建父工程 三 创建公共工具类 三 创建开发项目 四 创建开发项目子模块 五 运行项目 鉴于最近学习一个分布式项目的开发，讲一下关于使用IntelliJ IDEA基于Maven创...



williamHappy (<http://blog.csdn.net/williamHappy>) 2017-01-12 12:40 17946

maven配置文件中modules的作用 (/u010931123/article/details/68066326)

modules 从字面意思来说，module就是模块，而pom.xml中的modules也正是这个意思，用来管理同个项目中的各个模块；如果maven用的比较简单，或者说项目的模块在pom.xml...



u010931123 (<http://blog.csdn.net/u010931123>) 2017-03-29 16:22 2065

Maven多模块项目管理小结 (/whuslei/article/details/7989102)

题记 最近刚完成一个用Maven构建的Web项目，看了一些Maven方面的书，比如《maven实战》，但还是对Maven多模块项目理解得不清晰，所以花了一点时间好好研究了下，现分享如下。 问题 ...



whuslei (<http://blog.csdn.net/whuslei>) 2012-09-19 16:36 38692

关于Maven pom.xml中的元素modules、parent、properties以及import讲解推荐博客 (/smile__you/article/details/52072456)

maven讲解



Smile__you (http://blog.csdn.net/Smile__you) 2016-07-30 13:53 4624

从netty项目组织入门maven的多modules模块配置 (/hh544/article/details/43704221)

Netty是一套提供异步的、事件驱动的网络应用程序框架，同时也是工具包。我们可以将它作为项目的核心框架，同时也可以用它提供的部分功能来对项目进行支持，因此项目功能的模块化就显得很重要，这也是我们平时做...



hh544 (<http://blog.csdn.net/hh544>) 2015-02-10 15:58 1777



内容举报



返回顶部

Maven实战（三）——多模块项目的POM重构(转)

(/linfeng1991/article/details/46893771)

Maven实战（三）——多模块项目的POM重构


 linfeng1991 (http://blog.csdn.net/linfeng1991)

2015-07-15 15:15


 163

Maven最佳实践：划分模块 (/u010102162/article/details/40153103)

“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，m...


 u010102162 (http://blog.csdn.net/u010102162)

2014-10-16 18:57


 309

Maven最佳实践：划分模块 (/a1179785335/article/details/43270845)

转载地址：http://juvenshun.iteye.com/blog/305865 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...

 a1179785335 (http://blog.csdn.net/a1179785335)

2015-01-29 14:27

 167

Maven实战（三）——多模块项目的POM重构 (/project11/article/details/46695001)

在本专栏的上一篇文章POM重构之增还是删中，我们讨论了一些简单实用的POM重构技巧，包括重构的前提——持续集成，以及如何通过添加或者删除内容来提高POM的可读性和构建的稳定性。但在实际的项目中...

 project11 (http://blog.csdn.net/project11)

2015-06-30 13:46

 638

Maven最佳实践：划分模块 (/zhao1949/article/details/53887175)

http://juvenshun.iteye.com/blog/305865 ***** “分天下为三十六郡，郡置守，尉，监...

 zhao1949 (http://blog.csdn.net/zhao1949)

2016-12-26 16:29

 86

Maven最佳实践：划分模块 (/lwjshuai/article/details/76974515)

“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...

 lwjshuai (http://blog.csdn.net/lwjshuai)

2017-08-09 10:50

 57

Maven最佳实践：划分模块 (/yydcj/article/details/8698222)

“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...



内容举报



返回顶部



yydcj (<http://blog.csdn.net/yydcj>) 2013-03-20 20:31 803

Maven最佳实践：划分模块 (/lxyhenpiaoliang/article/details/68063840)

转载自：<http://juvenshun.iteye.com/blog/305865> “分天下为三十六郡，郡置守，尉，监” —— 《史记·秦始皇本纪》
所有用Mave...



lxyhenpiaoliang (<http://blog.csdn.net/lxyhenpiaoliang>) 2017-03-29 13:38 75

Maven最佳实践：划分模块 (/caolaosanahnu/article/details/7926702)

<http://juvenshun.iteye.com/blog/305865> “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...



caolaosanahnu (<http://blog.csdn.net/caolaosanahnu>) 2012-08-30 20:18 397



内容举报



返回顶部