maven学习笔记

6/14/2010 8:08 PM

# 1、Pom.xml中的配置

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.apache.servicemix</groupId>

<artifactId>parent</artifactId>

<version>3.3.1</version>

</parent>

<groupId>org.apache.servicemix</groupId>

<artifactId>serviceengines</artifactId>

<version>3.3.1</version>

<name>ServiceMix :: Service Engines</name>

<packaging>pom</packaging>

<modules>

<module>servicemix-lwcontainer</module>

</modules>

<build>

<plugins>

<plugin>

<groupId>org.apache.servicemix</groupId>

<artifactId>servicemix-build</artifactId>

<version>1.0</version>

<executions>

<execution>

<goals>

<goal>spring.schemas</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

这个模块缺少这个version节点，或找不到合适的servicemix-build版本

# 2、资源库中pom出现错误

资源库中的pom文件是这样的：

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>301 Moved Permanently</title>

</head><body>

<h1>Moved Permanently</h1>

<p>The document has moved <a href="http://download.java.net/maven/1/org.apache.servicemix/poms/servicemix-pom-2.pom">here</a>.</p>

<hr>

<address>Apache Server at maven-repository.dev.java.net Port 443</address>

</body></html>

就要手动去下载正确的pom文件放在资源库这个位置

# 3、较好的远程资源库

如果下载不到jar包或者下载的有有问题，请换用较好的远程资源库：

修改maven/conf/ settings.xml中的mirror

较好的有：

<mirror>

<id>ibiblio.org</id>

<name>ibiblio Mirror of http://mirrors.ibiblio.org/pub/mirrors/maven2</name>

<url>http://mirrors.ibiblio.org/pub/mirrors/maven2</url>

<mirrorOf>central</mirrorOf>

<!-- United States, North Carolina -->

</mirror>

# 4、pom.xml配置覆盖

项目中的pom.xml文件中的设置（比如：代码库、插件命令的选项）会覆盖{maven\_home}/conf/setting.xml中的或者maven默认的设置，所以遇到与此相关的问题，尝试修改pom.xml的配置（如：修改pom.xml文件中的远程代码库地址，修改插件的选项设置）。

# 5、Reason: Not a v4.0.0 POM. for project

出现这个错误，请在mvn命令加 –e 参数，打印出详细错误信息。将会看出是maven自身的插件的pom.xml文件不对（可能是网络原因），删除重新运行mvn进行下载。

# 6、编译大项目遇到困难

在编译整个项目的时候遇到非常难编译的子项目时，可以到这个子项目文件夹下去单独编译此子项目。

# 7、“javac: 无效的目标版本: 1.6”

在命令行下构建一个工程时，出现“javac: 无效的目标版本: 1.6“的提示，构建失败。

解决：试着把javac版本提高到1.6的水平。

# 8、Skipping Tests

To skip running the tests for a particular project, set the skipTests property to true.

|  |
| --- |
|  <project>     [...]     <build>       <plugins>         <plugin>           <groupId>org.apache.maven.plugins</groupId>           <artifactId>maven-surefire-plugin</artifactId>           <version>2.4.2</version>           <configuration>             <skipTests>true</skipTests>           </configuration>         </plugin>       </plugins>     </build>     [...]   </project> |

You can also skip the tests via command line by executing the following command:

***>>> mvn install -DskipTests***

If you absolutely must, you can also use the maven.test.skip property to skip compiling the tests. maven.test.skip is honored by Surefire and the Compiler Plugin.

***>>> mvn install -Dmaven.test.skip=true***

# 9、Maven plugin中的lifecycle、phase、goal、mojo概念及作用的理解

   首先，说些题外话，maven的plugin真的很容易写，很多时候，我们只是被plugin这个词吓倒了，总以为插件这玩意，是专家才能写的，我 maven都没精通，怎么写得出自己的plugin呢，其实不然，起码在maven中，写一个自己的plugin还是非常简单的，其它软件的插件，要看情 况，有些的确是要天才级人物才写得出，有一些呢，也无非是用别人做的傻瓜程序，可以轻松做出来，但是，有决心做，绝大数事情我们是做得到的！

      要写自己的maven plugin的话，lifecycle与phase与goal与mojo的概念是一定要理解的，下面是我自己的一些见解

**lifecycle：**生命周期，这是maven最高级别的的控制单元，它是一系列的phase组成，也就是说， 一个生命周期，就是一个大任务的总称，不管它里面分成多少个子任务，反正就是运行一个lifecycle，就是交待了一个任务，运行完后，就得到了一个结 果，中间的过程，是phase完成的，自己可以定义自己的lifecycle，包含自己想要的phase

常见的lifecycle有 | clean | package ear | pageage jar | package war | site等等

**phase：**可以理解为任务单元，lifecycle是总任务，phase就是总任务分出来的一个个子任务，但 是这些子任务是被规格化的，它可以同时被多个lifecycle所包含，一个lifecycle可以包含任意个phase，phase的执行是按顺序的， 一个phase可以绑定很多个goal，至少为一个，没有goal的phase是没有意义的

下面就是一些default lifecycle的phase

validate   
initialize   
generate-sources   
process-sources   
generate-resources   
process-resources  
compile compile   
process-classes   
generate-test-sources   
process-test-sources   
generate-test-resources   
process-test-resources   
test-compile   
process-test-classes   
test   
prepare-package  
package   
pre-integration-test   
integration-test   
post-integration-test   
verify   
install   
deploy

**goal:** 这是执行任务的最小单元，它可以绑定到任意个phase中，一个phase有一个或多个 goal，goal也是按顺序执行的，一个phase被执行时，绑定到phase里的goal会按绑定的时间被顺序执行，不管phase己经绑定了多少个 goal，你自己定义的goal都可以继续绑到phase中

**mojo:** lifecycle与phase与goal都是概念上的东西，mojo才是做具体事情的，可以简单理解mojo为goal的实现类，它继承于 AbstractMojo，有一个execute方法，goal等的定义都是通过在mojo里定义一些注释的anotation来实现的，maven会在 打包时，自动根据这些anotation生成一些xml文件，放在plugin的jar包里

      抛开mojo不讲，lifecycle与phase与goal就是级别的大小问题，引用必须是从高级引用下级（goal绑定到phase，也可理间为 phase引用goal，只是在具体绑定时，不会phase定义引用哪些goal，但是执行是，却是phase调用绑定到它那的goal），也不能跨级引 用，如lifecycle可以引用任意的phase，不同lifecycle可以同时引用相同的phase，lifecycle不能跨级引用goal。 goal会绑定到任意的phase中，也就是说不同的phase可以同时引用相同的goal，所以goal可以在一个lifecycle里被重复执行 哦，goal自然也不能说绑定到lifecycle中，它们三者的关系可以用公司里的 总领导，组领导，与职员的关系来解释

# 10、pom.xml中的变量${property}

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.6</source>  <target>1.6</target>  <compilerVersion>1.6</compilerVersion>  <executable>${jdk16\_home}/bin/javac</executable>  <fork>true</fork>  <verbose>true</verbose>  </configuration>  </plugin> |

pom.xml文件中的变量${jdk16\_home}可以在运行mvn命令时加入：

>>> mvn install -Dmaven.test.skip=true -e -Djdk16\_home=D:/dev\_space/java/JDK/jdk1.6.0\_14

或者配置文件中配置参数：

|  |
| --- |
|   <properties>       <property>         <name>test</name>         <value>some-value</value>       </property>   </properties>     ...   or     ...     <properties>       <test>some-value</test>   </properties> |

# 11、-DdownloadSources=true

下载jar相应的源代码jar

# 12、archiva Remote Repositories

archiva 添加 **R**emote Repositories 时在linux内，可能会报没权限写入配置文件/conf/archiva.xml，可以手工创建/conf文件夹，并chmod 授权 777.

# 13、忽略 已过时 警告

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.3.1</version>  <configuration>  <source>1.6</source>  <target>1.6</target>  **<deprecation />**  </configuration>  </plugin> |

在编译插件中添加**deprecation**标志是可以实现忽略“已过时”警告的。

但是在编译geronimo时其根配置pom.xml文件配置中有

|  |
| --- |
| <parent>  <groupId>org.apache.geronimo.genesis</groupId>  <artifactId>genesis-java5-flava</artifactId>  <version>2.0</version>  </parent> |

此父配置中含有编译插件配置，导致把以便插件放在哪个子pom.xml文件中都不起作用。

由此看来pom.xml文件的子配置不能覆盖父配置。。。

# 14、警告：sun.misc.CompoundEnumeration 是 Sun 的专用 API，可能会在未来版本中删除

此问题可以用 来解决

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.3.1</version>  <configuration>  <source>1.6</source>  <target>1.6</target>  <deprecation />  </configuration>  <dependencies>  <dependency>  <groupId>org.codehaus.plexus</groupId>  <artifactId>plexus-compiler-javac</artifactId>  <version>1.8.1</version>  </dependency>  </dependencies>  </plugin> |

但是这样就会出现另外的问题：***UTF-8字符映射问题***

把**编译级别调成1.5**可以解决此问题

# 15、类找不到

|  |
| --- |
| <dependency>  <groupId>org.codehaus.groovy</groupId>  <artifactId>groovy-all-minimal</artifactId>  </dependency> |

此依赖中有一类找不到，并且版本没有定义，也不好查找到底哪儿问题。

确实在现有的各版本中都找不到此类，但在groovy-all中可以找到，可能是jar包名写错，换用groovy-all，编译可以通过。

# 16、[WARNING] \*\*\* CHECKSUM FAILED – Checksum failed on download: local = ‘0f56ee033ef4b78d33722aceb71a09c7ed68183f’; remote = ‘ Downloading: https://maven-repository.dev.java.net/nonav/repository//org.apache/poms/apache-3.pom

下载错误不能读取zip文件。

That is some checksum coming from https://maven-repository.dev.java.net/nonav/repository, definitely can’t be right! The culprit lies in org.apache.axis2:axis2-parent:1.5.1:

[view source](http://www.ashlux.com/wordpress/2010/06/15/axis2-maven-and-java.net-woes-and-why-maven-sucks/#viewSource)

[print](http://www.ashlux.com/wordpress/2010/06/15/axis2-maven-and-java.net-woes-and-why-maven-sucks/#printSource)[?](http://www.ashlux.com/wordpress/2010/06/15/axis2-maven-and-java.net-woes-and-why-maven-sucks/#about)

01.<repository>

02.    <id>java.net</id>

03.    <url>[https://maven-repository.dev.java.net/nonav/repository/<](https://maven-repository.dev.java.net/nonav/repository/%3C);/url>

04.    <layout>legacy</layout>

05.    <snapshots>

06.        <enabled>false</enabled>

07.    </snapshots>

08.    <releases>

09.        <enabled>true</enabled>

10.    </releases>

11.</repository>

The problem is [Maven does not support permanent redirects (301)](http://jira.codehaus.org/browse/MNG-4428). There are several workarounds, from modifying your local repository to adding a mirror for java.net for your settings.xml, none of which I find appropriate since builds are unnecessarily broken for new developers. I chose the settings.xml since I think it’s the cleanest out of all of them:

[view source](http://www.ashlux.com/wordpress/2010/06/15/axis2-maven-and-java.net-woes-and-why-maven-sucks/#viewSource)

[print](http://www.ashlux.com/wordpress/2010/06/15/axis2-maven-and-java.net-woes-and-why-maven-sucks/#printSource)[?](http://www.ashlux.com/wordpress/2010/06/15/axis2-maven-and-java.net-woes-and-why-maven-sucks/#about)

01.<settings>

02.  ...

03.  <mirrors>

04.    ...

05.    <mirror>

06.      <id>java.net.2</id>

07.      <name>java.net Mirror of [http://repo1.maven.org/maven2/<](http://repo1.maven.org/maven2/%3C);/name>

08.      <url>[http://download.java.net/maven/2/<](http://download.java.net/maven/2/%3C);/url>

09.      <mirrorOf>java.net</mirrorOf>

10.    </mirror>

11.  </mirrors>

12.</settings>

在setting文件中添加mirror可以解决。

不知道maven选择远程库的顺序是什么样的？

# 17、Eclipse中Maven项目运行在tomcat上

添加插件到pom.xml中

|  |
| --- |
| <plugin>  <groupId>org.codehaus.mojo</groupId>  <artifactId>tomcat-maven-plugin</artifactId>  <version>1.1</version>  <configuration>  <path>${contextpath-name}</path>  </configuration>  </plugin> |

如果设置${contextpath-name}为myapp，则web应用的地址URL为http://localhost:8080/myapp

运行：在Run --> Run Configurations 里set the Goal --> **tomcat:run** (this is a default goal for Tomcat)。

## Why don't I have to install Tomcat with 2.x?

If you run **mvn integration-test**, the Maven Cargo plugin will download and install Tomcat and run the integration tests against it. If you want to see your project running in a server, you have two options:

1. Run **mvn jetty:run-war** - this will fire up a Jetty instance and it will be available at [http://localhost:8080](http://localhost:8080/).
2. Run **mvn cargo:start -Dcargo.wait=true** and your application will be deployed to Tomcat and available at <http://localhost:8081/applicationname-version>.

You can also add the [Maven 2 Tomcat Plugin](http://mojo.codehaus.org/tomcat-maven-plugin/introduction.html) to your pom.xml and use **mvn tomcat:run** and **mvn tomcat:run-war**.

|  |
| --- |
| <plugin>      <groupId>org.codehaus.mojo</groupId>      <artifactId>tomcat-maven-plugin</artifactId>      <configuration>          <path>/</path>      </configuration>  </plugin> |

## What if I already have Tomcat installed and I'd like to use it?

If you want to change [Cargo](http://cargo.codehaus.org/) to point to an existing instance of Tomcat, you should be able to change the bottom of pom.xml from:

|  |
| --- |
| <container>      <containerId>${cargo.container}</containerId>      <!--home>${cargo.container.home}</home-->      <zipUrlInstaller>          <url>${cargo.container.url}</url>          <installDir>${installDir}</installDir>      </zipUrlInstaller>  </container> |

To:

|  |
| --- |
| <container>      <containerId>${cargo.container}</containerId>      <home>${cargo.container.home}</home>  </container> |

You can also [deploy to a running instance of Tomcat with Cargo](http://cargo.codehaus.org/Deploying+to+a+running+container).