本文您将学到以下内容：

1、maven的安装部署  
2、maven原理解析  
3、nexus私服搭建  
4、jekins环境搭建  
5、jekins与maven集成  
6、cargo自动部署web应用  
7、maven插件使用  
......

1     maven安装

1.1   下载

下载apache-maven-3.2.1放到E:\maven\目录下

解压：

l  bin   Maven的运行脚本

l  boot  Maven自己的类装载器

l  conf  该目录下包含了全局行为定制文件setting.xml

l  lib   Maven运行时所需的类库

将E:\maven\apache-maven-3.2.1\bin添加到环境变量

检查是否安装成功：

小试Maven命令

|  |
| --- |
| mvn help:system |

该命令将会下载help插件并运行它，且打印出Java系统属性和环境变量

1.2   配置本地仓库目录

         默认路径是在用户目录下的.m2文件夹下，但这不安全，有可能会被删除。建议放到一个不会被系统删除的安全的目录下：

修改maven目录下的conf中的setting.xml（59行）

|  |
| --- |
| <localRepository>E:/maven/LocalRepository</localRepository> |

2     第一个maven项目

2.1   安装m2e插件（其他文档有，就不在赘述）

搜索m2eclipse：

安装与eclipse对应版本的m2e,我这里是juno版本，选的第一个

2.2   创建一个简单的eclipse工程

项目结构：

编写最简单的类文件：

|  |
| --- |
| package com.lbl.helloword;    /\*\*   \* Hello world!   \*   \*/  publicclass HelloWord {         public String sayHello(String name){        return "Hello : " + name ;     }    } |

|  |
| --- |
| package com.lbl.helloword;    import org.junit.Assert;  import org.junit.Test;      /\*\*   \* Unit test for simple App.   \*/  publicclass HelloWordTest {         @Test     publicvoid testSayHello(){        Assert.*assertEquals*("Hello : lisi",new HelloWord().sayHello("lisi"));     }    } |

pom.xml:

|  |
| --- |
| <projectxmlns=*"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/xsd/maven-4.0.0.xsd"*>    <modelVersion>4.0.0</modelVersion>      <groupId>com.lbl</groupId>    <artifactId>helloword</artifactId>    <version>0.0.1-SNAPSHOT</version>    <packaging>jar</packaging>      <name>helloword</name>    <url>http://maven.apache.org</url>      <properties>      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>    </properties>      <dependencies>      <dependency>        <groupId>junit</groupId>        <artifactId>junit</artifactId>        <version>4.7</version>        <scope>test</scope>      </dependency>    </dependencies>    </project> |

运行:

选择Maven build…输入

|  |
| --- |
| mvn clean install |

运行结束：

3     Maven依赖包查询地址

[http://mvnrepository.com](http://mvnrepository.com/)（好用）

<http://search.maven.org/>

4     聚合

可以在一个地方编译（不只是编译，打包，部署等都可以）其他项目

4.1   创建四个等待聚合的项目

在user-aggregation项目执行mvn cleanpackage即可一次性处理四个项目。

5     继承

         想上面那四个项目，pom.xml文件中内容好多都是重复的，而且为了使得整个系统都采用统一的依赖管理（依赖库版本统一），我们将使用继承。

         新建user-parent项目（实际上可以不建项目，只要pom.xml放到其他项目的同级目录下即可。只是由于eclipse是通过项目管理文件的，所以这里才新建项目）

user-parent项目的pom.xml文件：

|  |
| --- |
| <projectxmlns=*"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/xsd/maven-4.0.0.xsd"*>    <modelVersion>4.0.0</modelVersion>      <groupId>com.lbl</groupId>    <artifactId>user-parent</artifactId>    <version>0.0.1-SNAPSHOT</version>    <packaging>pom</packaging>      <url>http://maven.apache.org</url>      <properties>      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>      <junit.version>4.10</junit.version>    </properties>    <!-- 父项目dependencyManagement中定义的依赖，子项目中不会自动继承，需要使用dependency声明才可以 -->  <!-- 但是子项目中只需要写groupId和artifactId即可，子项目会去父项目中寻找 -->     <dependencyManagement>        <dependencies>           <dependency>              <groupId>junit</groupId>              <artifactId>junit</artifactId>              <version>${junit.version}</version>              <scope>test</scope>           </dependency>           <dependency>              <groupId>servletapi</groupId>              <artifactId>servletapi</artifactId>              <version>2.4</version>              <scope>provided</scope>           </dependency>             <dependency>              <groupId>javax.servlet.jsp</groupId>              <artifactId>jsp-api</artifactId>              <version>2.2</version>              <scope>provided</scope>           </dependency>             <dependency>              <groupId>jstl</groupId>              <artifactId>jstl</artifactId>              <version>1.2</version>           </dependency>             <dependency>              <groupId>taglibs</groupId>              <artifactId>standard</artifactId>              <version>1.1.1</version>           </dependency>        </dependencies>     </dependencyManagement>    </project> |

user-log项目的pom.xml文件修改为：

|  |
| --- |
| <projectxmlns=*"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/xsd/maven-4.0.0.xsd"*>    <modelVersion>4.0.0</modelVersion>  <!-- 将一些重复的信息去掉 -->    <artifactId>user-log</artifactId>    <packaging>jar</packaging>      <name>user-log</name>      <parent>    <groupId>com.lbl</groupId>      <artifactId>user-parent</artifactId>      <version>0.0.1-SNAPSHOT</version>  <!--     其实是继承的pom.xml文件 -->    <relativePath>../user-parent/pom.xml</relativePath>    </parent>      <!-- 由于使用了继承，这里只需要写groupId和artifactId即可 .  子项目可以覆盖父项目的信息-->    <dependencies>      <dependency>        <groupId>junit</groupId>        <artifactId>junit</artifactId>      </dependency>    </dependencies>    </project> |

再次执行聚合项目即可

6     nexus 私服

windows环境下下载nexus-2.8.0-05-bundle.zip（这个版本要使用jdk7）

解压到E:\maven\nexus-2.8.0-05-bundle

将  E:\maven\nexus-2.8.0-05-bundle\nexus-2.8.0-05\bin  添加到环境变量

修改E:\maven\nexus-2.8.0-05-bundle\nexus-2.8.0-05\bin\jsw\conf\wrapper.conf文件

15行修改为：

|  |
| --- |
| wrapper.java.command=D:\Program Files\Java\jdk1.7.0\_55\bin\java |

但是我在win7系统上安装失败，总是提示一下错误信息，未解决。

在window server 2003上安装成功

访问<http://127.0.0.1:8081/nexus>

点击login 用户名：admin，密码admin123

6.1   私服仓库简介

l  host的仓库。内部项目的发布仓库

l  proxy的仓库。从远程中央仓库中寻找数据的仓库

l  group仓库。组仓库用来方便开发人员进行设置的仓库

6.2   私服配置

6.2.1  更新索引

将type为proxy的Download remote Indexs设为TRUE

可以看到多了几个任务正在执行：

索引下载完毕后，

6.2.2  配置镜像

修改E:\maven\apache-maven-3.2.1\conf\settings.xml文件

|  |
| --- |
| <mirrors>           <mirror>        <id>central</id>        <mirrorOf>\*</mirrorOf><!—所有仓库都从这里获取，也可以写成centeal,nexus指id为centeal,nexus的仓库才从这里取 -->        <name>Human Readable Name for this Mirror.</name>        <url>http://192.168.123.190:8081/nexus/content/groups/public/</url>      </mirror>    </mirrors> |
| <servers>                                       <server>                                                 <id>nexus-snapshots</id>                                                 <username>admin</username>                                                 <password>admin123</password>                                       </server>        </servers> |

6.2.3  让本机所有maven项目都去私服下载

修改E:\maven\apache-maven-3.2.1\conf\settings.xml

|  |
| --- |
| <profiles>           <profile>        <repositories>          <repository>            <id>central</id>            <name>Central</name>            <url>http://central</url> <!-- 配置镜像以后这个url就没有意义了 -->             <!-- 开启对releases版本的依赖 -->                       <releases><enabled>true</enabled></releases>           <!-- 开启对snapshots版本的依赖 -->                       <snapshots><enabled>true</enabled></snapshots>          </repository>        </repositories>      </profile>  </profiles> |

这样就不用在具体项目中配置repositories了。但是引入koala的时候还是得写，目前还不知为什么

parent项目的pom.xml显得很干净：

6.2.4  配置eclipse使用刚修改的settings.xml文件

6.2.5  验证

将poi的pom.xml信息拷贝到parent项目的pom.xml文件中

在user-log项目中继承

|  |
| --- |
| <dependency>        <groupId>org.apache.poi</groupId>        <artifactId>poi</artifactId>     </dependency> |

在maven的控制台可以看到他没有去中心仓库下载而是在我配置的私服中去下载

进入私服服务器的nexus安装目录C:\Program Files\nexus-2.8.0-05-bundle\sonatype-work\nexus\storage\central\org\apache\，可以看到poi已经下载到私服

在nexus页面搜索poi得到如下结果（不知为什么，在下载之前查不到，必须要在项目中使用改一次才可以）：

6.3   第三方的jar上传到nexus上面

查看上传的ojdbc14

在项目中就可以这样引用了

|  |
| --- |
| <dependency>             <groupId>com.oracle</groupId>             <artifactId>ojdbc</artifactId>             <version>10.2.0.1.0</version>           </dependency> |

7     Jenkins环境搭建（持续继承部署CI）

http://www.linuxidc.com/Linux/2012-07/64182p2.htm

7.1   安装jenkins服务

下载jenkins.war，将其放倒 tomcat的webapps下，启动tomcat，进入jenkins的控制页面，非常简单：

将所有项目的svn都配置好

执行立即构建后，工作空间是这样的：

7.2   手动安装插件

有时自动安装插件失败，这是需要手动安装插件，插件下载地址：

<http://updates.jenkins-ci.org/download/plugins/>

依次安装一下插件：

analysis-core(findbugs要用到它)

findbugs

8     使用Cargo插件自动化部署web容器

|  |
| --- |
| <build>          <finalName>rps-core</finalName>          <plugins>           <plugin>              <groupId>org.codehaus.cargo</groupId>                  <artifactId>cargo-maven2-plugin</artifactId>                  <version>1.1.3</version>                <configuration>                  <container>  <!--                  tomcat7部署时有点问题 -->                     <containerId>tomcat6x</containerId>                     <type>remote</type>                  </container>                  <configuration>                     <type>runtime</type>                     <properties>                        <cargo.remote.username>tomcat</cargo.remote.username>                        <cargo.remote.password>tomcat</cargo.remote.password>>                        <cargo.tomcat.manager.url>http://192.168.123.232:10083/manager</cargo.tomcat.manager.url>                     </properties>                  </configuration>              </configuration>           </plugin>        </plugins>     </build> |

8.1   解决tomcat6自动部署失败，无法删除旧的项目

tomcat-users.xml添加用于部署项目的用户（用到了tomcat）：

|  |
| --- |
| <role rolename="tomcat"/>    <role rolename="role1"/>    <user username="tomcat" password="tomcat" roles="manager,tomcat,admin"/>    <user username="both" password="tomcat" roles="tomcat,role1"/>    <user username="role1" password="tomcat" roles="role1"/> |

Conf/ context.xml相修改，Context节点添加以下内容：

|  |
| --- |
| <Context path="/rps" reloadable="false" docBase="E:\apache-tomcat-6.0.41-maven\webapps\rps" autoDeploy="false"                              antiResourceLocking="true" antiJARLocking="true"> |

8.2   解决一个机器侠配置多个tomcat问题

如果没有配置CATALINA\_HOME环境变量，那么直接解压多个tomcat即可。

但如果有一个tomcat由于某种原因添加到了CATALINA\_HOME环境变量，那么其他的tomcat解压后要修改bin目录下catalina.bat和startup.bat文件：

直接将所有%CATALINA\_HOME%修改为其他名称，比如：%CATALINA\_HOME2%

修改后的tomcat在本文件对应目录下

9     项目发布到私服

将releases和snapshots设置为允许发布

在parent项目的pom.xml文件中添加【发布的配置一般在项目中配置，而不再sesstings中配置】

|  |
| --- |
| <distributionManagement>    <repository>        <id>nexus-releases</id>        <name>User Project Releases</name>       <url>http://192.168.123.190:8081/nexus/content/repositories/releases/</url>    </repository>    <snapshotRepository>        <id>nexus-snapshots</id>        <name>User Project Snapshots</name>       <url>http://192.168.123.190:8081/nexus/content/repositories/snapshots/</url>    </snapshotRepository>    </distributionManagement> |

在maven的sessting.xml文件中配置用户信息（用户也可以用deployment，密码是deployment123。这个用户只能用于发布）

|  |
| --- |
| <server>                                                 <id>nexus-snapshots</id>                                                 <username>admin</username>                                                 <password>admin123</password>                                       </server>                                         <server>                                                 <id>nexus-releases</id>                                                 <username>admin</username>                                                 <password>admin123</password>                                       </server> |

注意：sessting.xml配置的server的id要和parent项目的pom.xml文件中注册的id一致，否则报错：没有权限

将之前聚合的信息添加到user-parent项目中（user-aggregation用户到了，可以删除）：

|  |
| --- |
| <modules>        <module>../user-core</module>        <module>../user-dao</module>        <module>../user-log</module>        <module>../user-service</module>     </modules> |

至此，user-parent项目pom.xml文件全部内容：

|  |
| --- |
| <projectxmlns=*"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/xsd/maven-4.0.0.xsd"*>    <modelVersion>4.0.0</modelVersion>      <groupId>com.lbl</groupId>    <artifactId>user-parent</artifactId>    <version>0.0.1-SNAPSHOT</version>    <packaging>pom</packaging>      <url>http://maven.apache.org</url>      <modules>        <module>../user-core</module>        <module>../user-dao</module>        <module>../user-log</module>        <module>../user-service</module>     </modules>      <distributionManagement>    <repository>        <id>nexus-releases</id>        <name>User Project Releases</name>       <url>http://192.168.123.190:8081/nexus/content/repositories/releases/</url>    </repository>    <snapshotRepository>        <id>nexus-snapshots</id>        <name>User Project Snapshots</name>       <url>http://192.168.123.190:8081/nexus/content/repositories/snapshots/</url>    </snapshotRepository>    </distributionManagement>      <properties>      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>      <junit.version>4.8.1</junit.version>    </properties>    <!-- 父项目dependencyManagement中定义的依赖，子项目中不会自动继承，需要使用dependency声明才可以 -->  <!-- 但是子项目中只需要写groupId和artifactId即可，子项目会去父项目中寻找 -->     <dependencyManagement>        <dependencies>           <dependency>              <groupId>junit</groupId>              <artifactId>junit</artifactId>              <version>${junit.version}</version>              <scope>test</scope>           </dependency>           <dependency>              <groupId>servletapi</groupId>              <artifactId>servletapi</artifactId>              <version>2.4</version>              <scope>provided</scope>           </dependency>             <dependency>              <groupId>javax.servlet.jsp</groupId>              <artifactId>jsp-api</artifactId>              <version>2.2</version>              <scope>provided</scope>           </dependency>             <dependency>              <groupId>jstl</groupId>              <artifactId>jstl</artifactId>              <version>1.2</version>           </dependency>             <dependency>              <groupId>taglibs</groupId>              <artifactId>standard</artifactId>              <version>1.1.1</version>           </dependency>             <dependency>              <groupId>org.apache.poi</groupId>              <artifactId>poi</artifactId>              <version>3.9</version>           </dependency>           <dependency>             <groupId>com.oracle</groupId>             <artifactId>ojdbc</artifactId>             <version>10.2.0.1.0</version>           </dependency>        </dependencies>     </dependencyManagement>    </project> |

运行  mvn clean deploy

控制台输出：

结果查看：

10          插件使用

访问[http://maven.apache.org](http://maven.apache.org/)，点击左侧plugins，列出maven自带的插件

maven的插件很好用，例子参数说明很清晰。但一些其他插件文档不规范，有时只能看源码来使用了。

下面举两个例子：

10.1      source插件

在user-parent项目中pom.xml添加

|  |
| --- |
| <build>        <pluginManagement>           <plugins>              <plugin>                  <groupId>org.apache.maven.plugins</groupId>                  <artifactId>maven-source-plugin</artifactId>                  <version>2.2.1</version>                    <!-- 在哪个生命周期内执行。当然也可以手动去执行，不过麻烦 -->                  <executions>                     <execution>                        <phase>package</phase><!--在package阶段执行 -->                        <goals><goal>jar-no-fork</goal></goals>执行的是插件的jar-no-fork目标                     </execution>                 </executions>              </plugin>           </plugins>        </pluginManagement>     </build> |

在user-log项目中pom.xml添加

|  |
| --- |
| <build>        <plugins>           <plugin>              <groupId>org.apache.maven.plugins</groupId>              <artifactId>maven-source-plugin</artifactId>           </plugin>        </plugins>     </build> |

执行 clean package 命令，源码打包了：

10.2      sql-maven-plugin插件

修改parent下面中pom.xml添加：

|  |
| --- |
| <plugin>                   <groupId>org.codehaus.mojo</groupId>                   <artifactId>sql-maven-plugin</artifactId>                   <version>1.5</version>                     <dependencies>                     <!-- 插件的依赖specify the dependentjdbc driver here -->                     <dependency>                       <groupId>mysql</groupId>                       <artifactId>mysql-connector-java</artifactId>                       <version>5.1.18</version>                     </dependency>                   </dependencies>                     <!-- common configuration shared by all executions -->                   <configuration>                     <driver>com.mysql.jdbc.Driver</driver>                     <url>jdbc:mysql://localhost:3306/数据库名</url>                     <username> username </username>                     <password> password </password>                     <sqlCommand>                     alter table id\_gen add adf int                     </sqlCommand>                   </configuration>                     <executions>                     <execution>                       <phase>package</phase>                       <goals><goal>execute</goal></goals>                     </execution>                   </executions>                 </plugin> |

在user-log下面中pom.xml添加

|  |
| --- |
| <plugin>              <groupId>org.codehaus.mojo</groupId>                <artifactId>sql-maven-plugin</artifactId>           </plugin> |

执行clean package 后会看到表中多了一列adf

11          附录

11.1      配置一下maven的代理服务器

访问nexus 服务器[http://192.168.123.190:8081/nexus/](http://192.168.123.190:8081/nexus/" \t "_blank)

在左侧菜单找到如图：

点击查看右边有很多选项，找到这里

添加你的代理服务器就可以了。

11.2     Tomcat项目部署方式

11.2.1 **一、静态部署**

**1、直接将web项目文件件拷贝到webapps 目录中**  
Tomcat的Webapps目录是Tomcat默认的应用目录，当服务器启动时，会加载所有这个目录下的应用。所以可以将JSP程序打包成一个war包放在目录下，服务器会自动解开这个war包，并在这个目录下生成一个同名的文件夹。一个war包就是有特性格式的jar包，它是将一个web程序的所有内容进行压缩得到。具体如何打包，可以使用许多开发工具的IDE环境，如Eclipse等。也可以用cmd命令：jar -cvf mywar.war myweb  
webapps这个默认的应用目录也是可以改变。打开Tomcat的conf目录下的server.xml文件，找到下面内容：

<Hostname="localhost" appBase="webapps"

unpackWARs="true"autoDeploy="true"

xmlValidation="false"xmlNamespaceAware="false">

将appBase修改即可。  
**2、在server.xml中指定**   
在Tomcat的配置文件中，一个Web应用就是一个特定的Context，可以通过在server.xml中新建Context里部署一个JSP应用程序。打开server.xml文件，在Host标签内建一个Context，内容如下。

在tomcat中的conf目录中，在server.xml中的，<host/>节点中添加：  
<Context path="/hello"docBase="D:\ workspace\hello\WebRoot"debug="0"privileged="true">  
< /Context>

或者

<Contextpath="/myapp"reloadable="true"docBase="D:\myapp"workDir="D:\myapp\work"/>

或者

<Contextpath="/sms4"docBase="D:\workspace\sms4\WebRoot"/>

说明：

path是虚拟路径；

docBase是应用程序的物理路径；

workDir是这个应用的工作目录，存放运行时生成的与这个应用相关的文件；

debug则是设定debuglevel, 0表示提供最少的信息，9表示提供最多的信息

privileged设置为true的时候，才允许Tomcat的Web应用使用容器内的Servlet

reloadable如果为true，则tomcat会自动检测应用程序的/WEB-INF/lib和/WEB-INF/classes目录的变化，自动装载新的应用程序，可以在不重起tomcat的情况下改变应用程序，实现热部署

antiResourceLocking和antiJARLocking热部署是需要配置的参数，默认false避免更新了某个webapp，有时候Tomcat并不能把旧的webapp完全删除，通常会留下WEB-INF/lib下的某个jar包，必须关闭Tomcat才能删除，这就导致自动部署失败。设置为true，Tomcat在运行对应的webapp时，会把相应的源文件和jar文件复制到一个临时目录里。  
**3、创建一个Context文件**在conf目录中，新建Catalina＼localhost目录，在该目录中新建一个xml文件，名字不可以随意取，要和path后的那个名字一致，按照下边这个path的配置，xml的名字应该就应该是hello（hello.xml），该xml文件的内容为：

<Contextpath="/hello" docBase="E:\workspace\hello\WebRoot"debug="0" privileged="true"></Context>

tomcat自带例子如下：

<ContextdocBase="${catalina.home}/server/webapps/host-manager"

privileged="true"antiResourceLocking="false" antiJARLocking="false">

</Context>

这个例子是tomcat自带的，编辑的内容实际上和第二种方式是一样的，其中这***xml文件名字就是访问路径，这样可以隐藏应用的真实名字。***

**4、注意：**

删除一个Web应用同时也要删除webapps下相应的文件夹和server.xml中相应的Context，还要将Tomcat的conf\catalina\localhost目录下相应的xml文件删除，否则Tomcat仍会去配置并加载。。。

11.2.2 **二 动态部署**

登陆tomcat管理控制台：[http://localhost:8080/](http://localhost:8080/" \t "_blank)，输入用户名和密码后便可管理应用并动态发布。

在ContextPath(option):中输入/yourwebname，这代表你的应用的访问地址。

XML Configration file URL中要指定一个xml文件，比如我们在F:\下建立一个hmcx.xml文件，内容如下：<Contextreloadable="false" />其中docBase不用写了，因为在下一个文本框中填入。或者更简单点，这个文本框什么都不填，在WAR orDirectory URL:中键入F:\hmcx即可，然后点击Deploy按钮，上面就可以看到了web应用程序，名字就Context Path(option):中的名字。

如果部署.war文件还有更加简单的方式，下面还有个Select WAR file uploae点击浏览选择.war文件，然后点击Deploy也可以。