

Maven 第二天

一、回顾

1.什么是 maven?

它是一个软件开发的工具，主要管理的工作是：依赖管理，项目构建

2.使用 maven 的好处？

能够集中管理 jar 包，提供一键构建

3.maven 的安装及配置

配置：系统变量新增 MAVEN_HOME,PATH 路径配置

本地仓库：<localRepository>

运行:mvn -v

4.常用的 maven 命令

clean,compile,test,package,install,deploy

5.maven 工程是具有一定的目录结构

src

main

java(程序主要代码)

resources (配置文件)

webapps

WEB-INF

web.xml

test

java (测试代码)

resources (测试的配置文件)

pom.xml (Maven 配置文件)

6.eclipse 工具下的 maven 工程开发

1.windows->Preferences->Maven->Installations 选择 add 增加 Maven 版本

2.windows->Preferences->Maven->User Settings 选择 settings.xml 文件，点击 Update Settings

3.Window-->show view-->other-->maven Repositories-->右键本地仓库(重建索引)

7.在 pom.xml 文件中如何引入坐标

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.5</version>
  <scope>provided</scope>
</dependency>
```

解决 jdk 版本默认 1.5 的问题

```
<build>
  <finalName>crmssh28</finalName>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-
plugin</artifactId>
        <version>3.2</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
          <encoding>UTF-8</encoding>
```

```

        <showWarnings>true</showWarnings>

    </configuration>

</plugin>

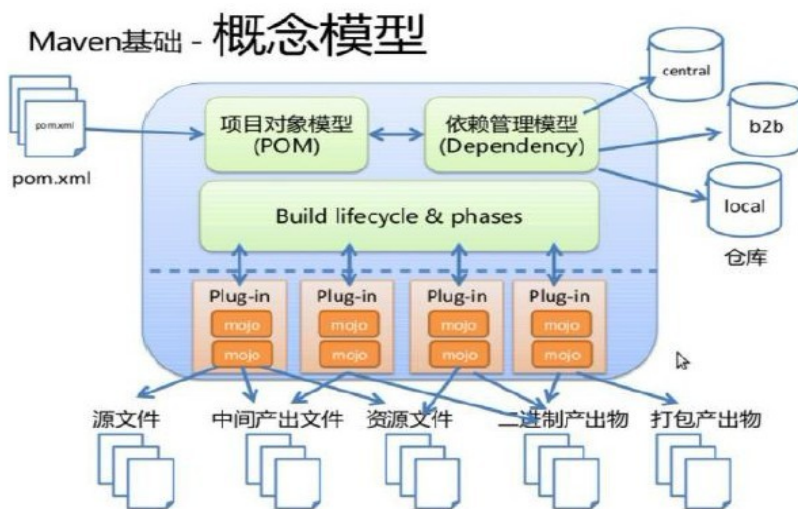
</plugins>

</pluginManagement>

</build>

```

8.总结



二、Maven 工程的拆分与聚合（重点）

一个完整的早期开发好的 crm 项目，现在要使用 maven 工程对它进行拆分，这时候就可以将 dao 拆解出来形成表现独立的工程，同样 service, action 也都这样拆分

- ▶ crmssh28
- ▶ crmssh28_dao
- ▶ crmssh28_service
- ▶ crmssh28_web

工程拆分之后，将来还要聚合（聚合就是将拆分的工程进一步组合在一起，又形成一个完整的项目）

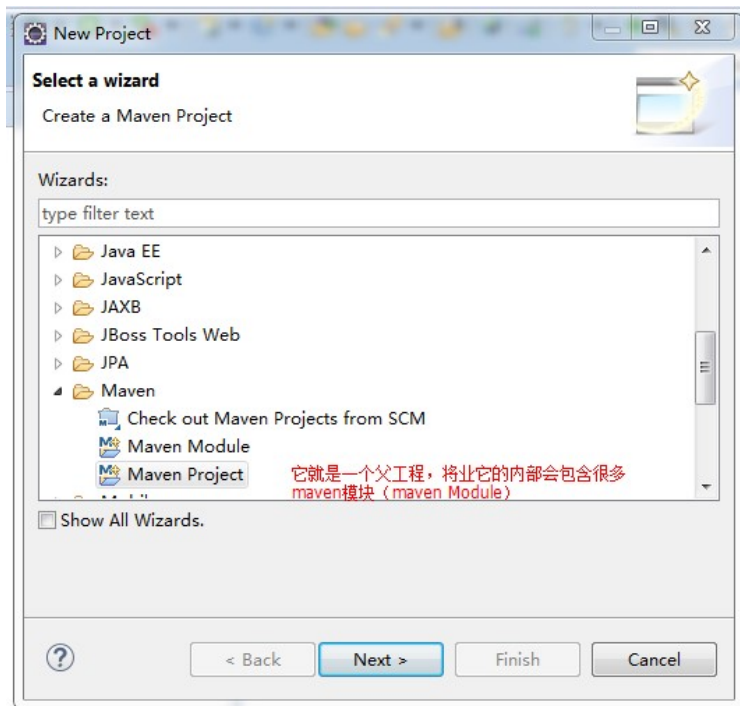
为了达到聚合的目标，所以今天会引入

父工程（maven project）

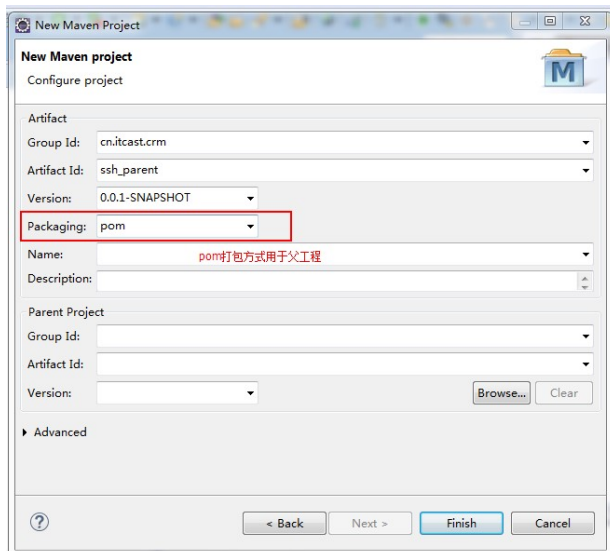
子模块(maven module) dao ,service, web

开发步骤：

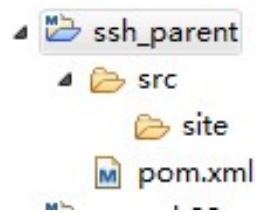
1. 创建一个 maven 父工程



点下一步：

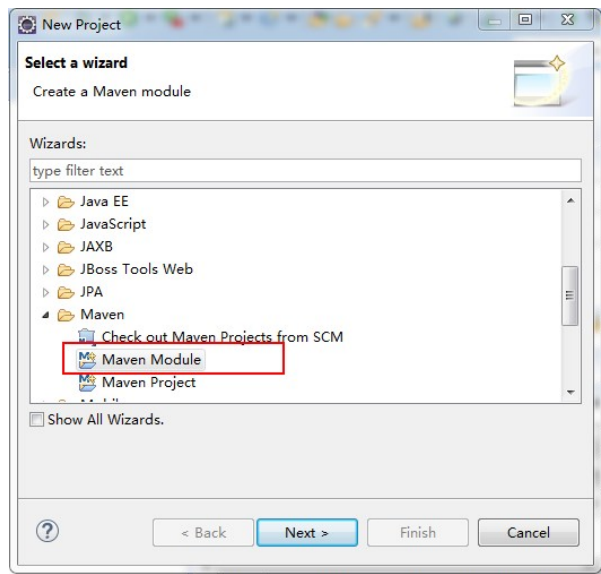


创建后的父工程如下：

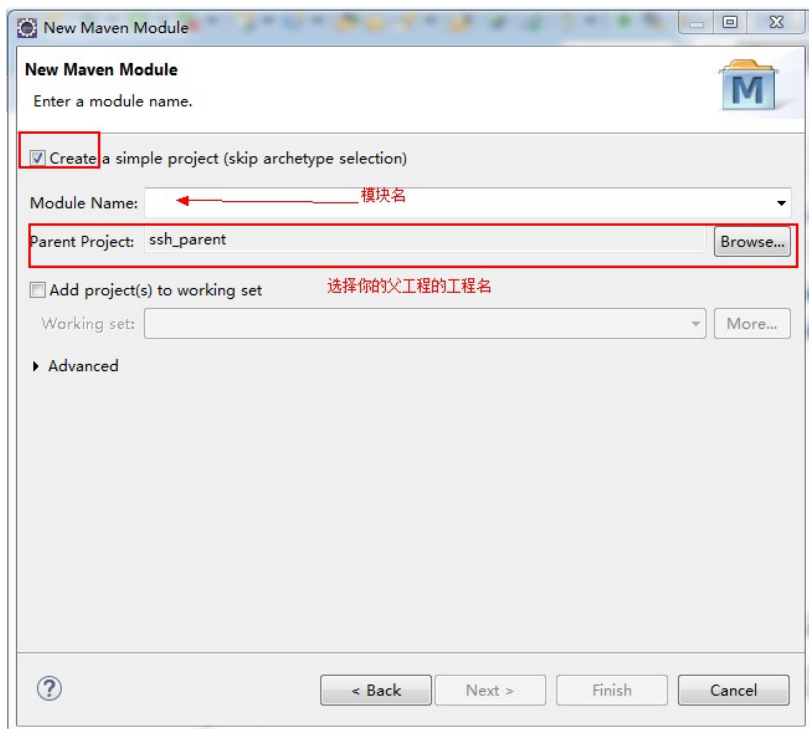


从它的目录结构可以看出，父工程本身不写代码，它里面有一个 pom.xml 文件，这个文件可以将多个子模块中通用的 jar 所对应的坐标，集中在父工程中配置，将来的子模块就可以不需要在 pom.xml 中配置通用 jar 的坐标了

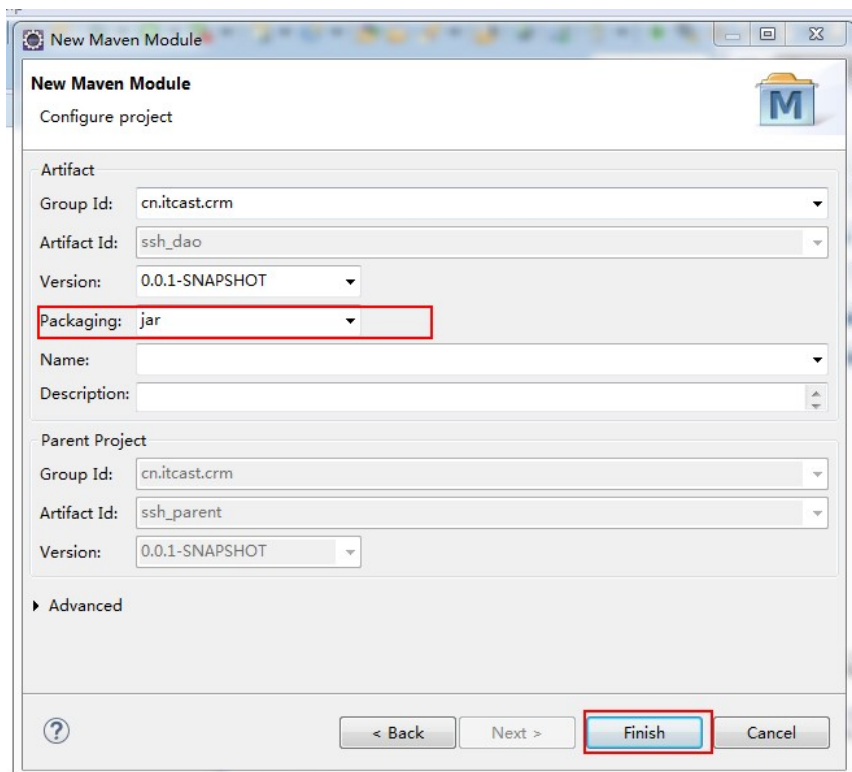
2. 如何创建这个父工程的一个子模块？



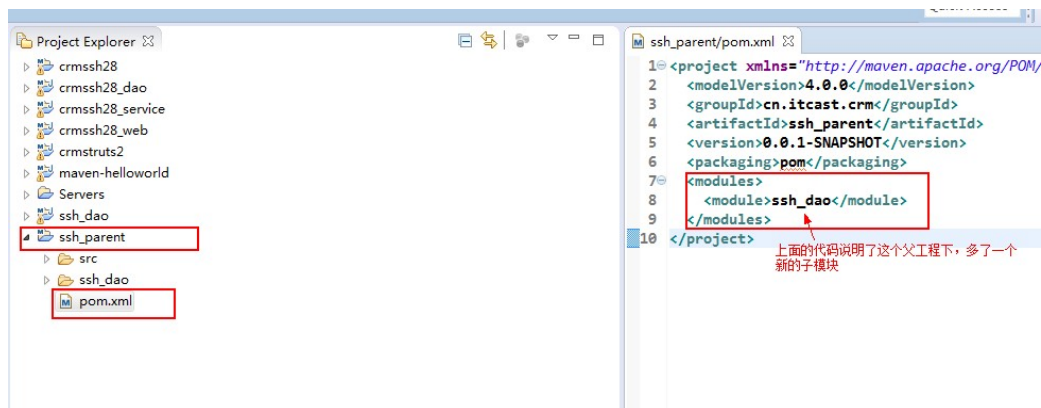
点 next,进入如下图：



点 next,进入如下图：



3. 再次查看父工程的 pom.xml 文件



4. 查看子模块的 pom.xml，发现多了一个 parent 结点

```
<parent>
  <groupId>cn.itcast.crm</groupId>
  <artifactId>ssh_parent</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</parent>
```

并且内部所包含的结点，其实就是父工程的坐标

坐标=groupId+artifactId+version

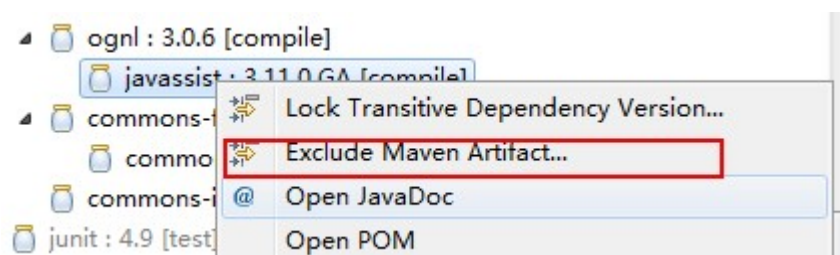
组织名 项目名 版本

三. 冲突问题的解决

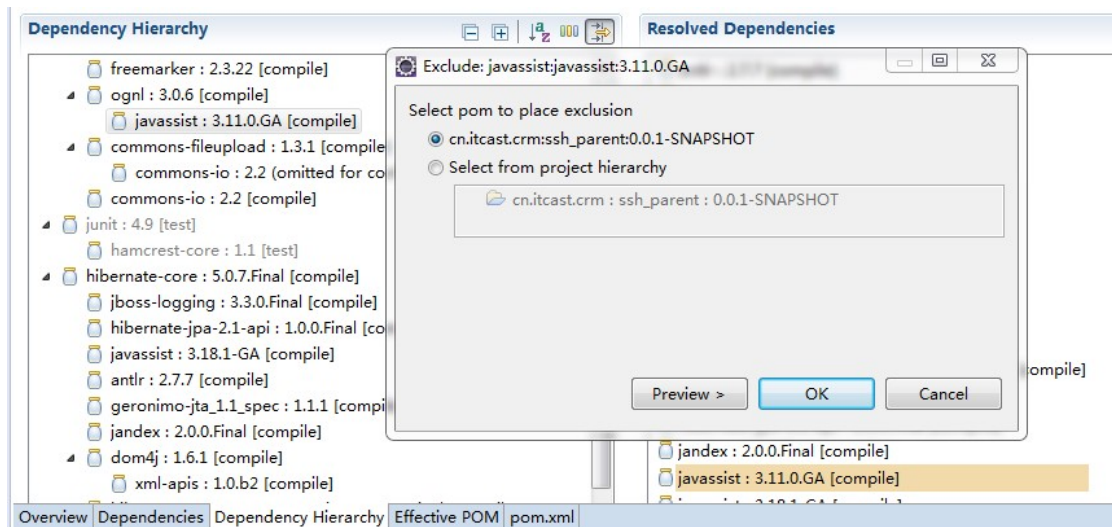
1.通过添加<exclusion>标签来解决冲突

在父工程中引入了 struts-core,hibernate-core，就发现 jar 包是有冲突的

Javassist 存在版本上冲突问题



进入下图：



背后的父工程的 pom.xml 文件中，添加的内容

```
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-core</artifactId>
  <version>2.3.24</version>
  <exclusions>
    <exclusion>
      <artifactId>javassist</artifactId>
      <groupId>javassist</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

2. 依赖调解原则：

maven 自动按照下边的原则调解：

✓ 1、第一声明者优先原则

在 pom 文件定义依赖，先声明的依赖为准。

测试：

如果将上边 struts-spring-plugins 和 spring-context 顺序颠倒，系统将导入 spring-beans-4.2.4。

分析：

由于 spring-context 在前边以 spring-context 依赖的 spring-beans-4.2.4 为准，所以最终 spring-beans-4.2.4 添加到了工程中。

✓ 2、路径近者优先原则

例如：A 依赖 spring-beans-4.2.4，A 依赖 B 依赖 spring-beans-3.0.5，则 spring-beans-4.2.4 优先被依赖在 A 中，因为 spring-beans-4.2.4 相对 spring-beans-3.0.5 被 A 依赖的路径最近。

测试：

在本工程中的 pom 中加入 spring-beans-4.2.4 的依赖，根据路径近者优先原则，系统将导入 spring-beans-4.2.4：

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-beans</artifactId>
  <version>4.2.4.RELEASE</version>
</dependency>
```

2.使用版本锁定实现冲突解决

首先父工程中 pom.xml 文件添加：

```
<dependencyManagement>
  <dependencies>
    <!--这里锁定版本为4.2.4 -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>4.2.4.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>4.2.4.RELEASE</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

说明了哪个jar包用什么版本

```

<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    </dependency>
  </dependencies>

```

不需要提供版本信息，因为上面已经锁定版本

在使用坐标时，对于同一个框架，引入多次时，它的版本信息就会多次出现，所以可以借用常量的思想，将这些版本号提取出来，在需要用到时，直接写版本的常量名称就可以了。

```

<properties>
  <spring.version>4.2.4.RELEASE</spring.version>
  <struts2.version>2.3.24</struts2.version>
  <hibernate.version>5.0.7.Final</hibernate.version>
  <slf4j.version>1.6.6</slf4j.version>
  <log4j.version>1.2.12</log4j.version>
  <shiro.version>1.2.3</shiro.version>
  <cxfr.version>3.0.1</cxfr.version>
  <c3p0.version>0.9.1.2</c3p0.version>
  <mysql.version>5.1.6</mysql.version>
</properties>

```

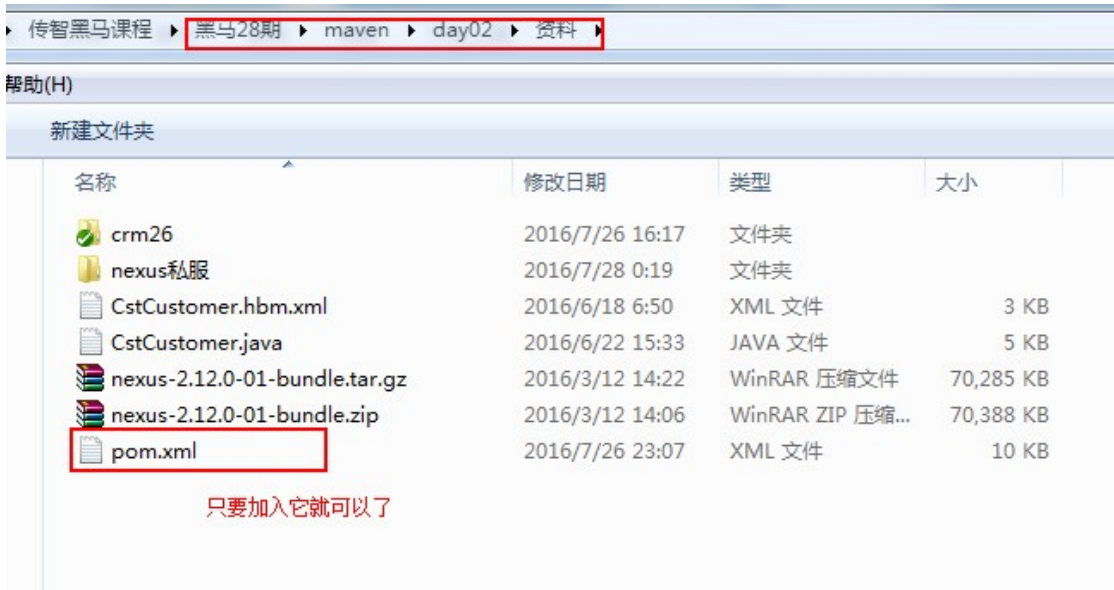
引用上面的常量

```

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
  </dependencies>

```

3.最终在 ssh_parent 的 pom.xml 中引入的坐标



三、依赖关系

依赖具有传递性，但不是无限传递的，传递的规则如下：

查看下图红色框内所示传递依赖范围：

直接依赖	传递依赖			
	compile	provided	runtime	test
compile	compile	-	runtime	-
provided	provided	provided	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

解决方法：

如果在依赖传递过程中，导致 jar 包丢失，我们的做法很简单，就是再导入一次坐标

四. 编写子模块（Server 模块举例）

1. 创建一个 maven module 项目

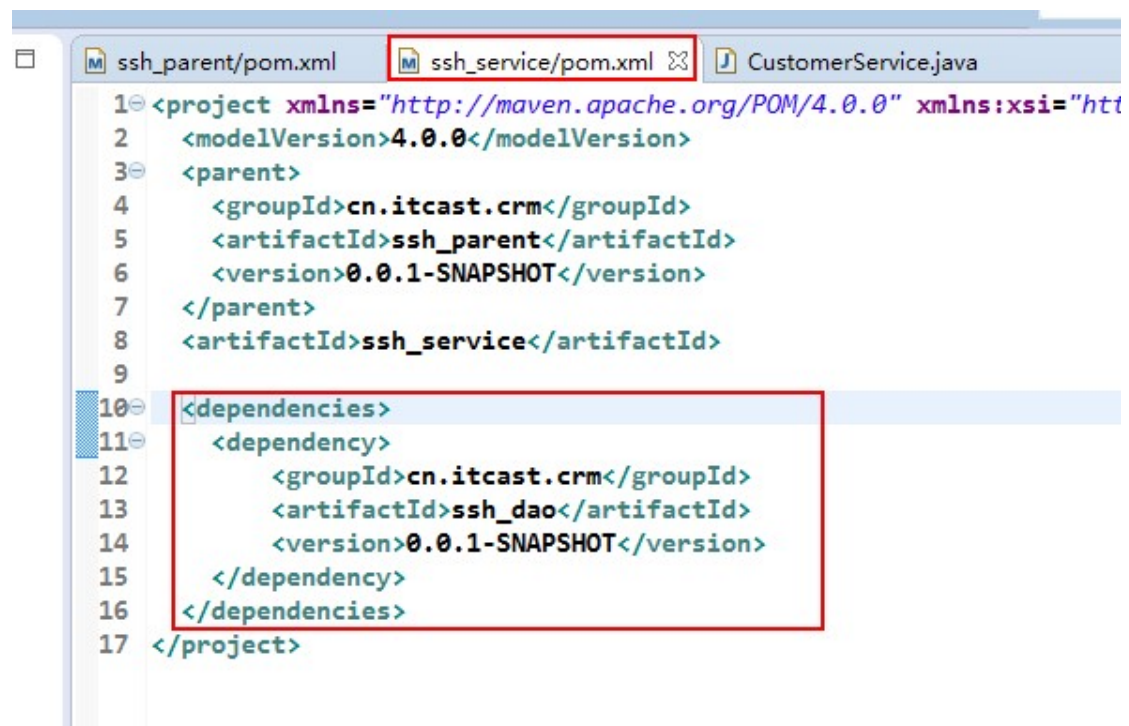
创建结束后，父工程中结构如下：



父工程的 pom.xml 文件如下

```
<modules>
  <module>ssh_dao</module>
  <module>ssh_service</module>
</modules>
```

2. 在 service 的 pom.xml 文件中引入 dao 的 jar 包



Web 层的子模块创建：

New Maven Module

Configure project

Artifact

Group Id: cn.itcast.crm

Artifact Id: ssh_web

Version: 0.0.1-SNAPSHOT

Packaging: war

Name:

Description:

Parent Project

Group Id: cn.itcast.crm

Artifact Id: ssh_parent

Version: 0.0.1-SNAPSHOT

Advanced

< Back Next > Finish Cancel

web.xml CustomerAction.java ssh_web/pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <parent>
5     <groupId>cn.itcast.crm</groupId>
6     <artifactId>ssh_parent</artifactId>
7     <version>0.0.1-SNAPSHOT</version>
8   </parent>
9   <artifactId>ssh_web</artifactId>
10  <packaging>war</packaging>
11  <dependencies>
12    <dependency>
13      <groupId>cn.itcast.crm</groupId>
14      <artifactId>ssh_service</artifactId>
15      <version>0.0.1-SNAPSHOT</version>
16    </dependency>
17  </dependencies>
18 </project>
```

五、私服搭建

下载 nexus

Nexus 是 Maven 仓库管理器，通过 nexus 可以搭建 maven 仓库，同时 nexus 还提供强大的仓库管理功能，构件搜索功能等。

下载 Nexus，下载地址：<http://www.sonatype.org/nexus/archived/>

2 Download a Distribution

Once you've selected a version in Step 1, you can download a distribution from the following list.

Download Nexus 2.12.0-01



NEXUS OSS (TGZ)

Checksums: MD5 SHA Signature: PGP

NEXUS OSS (ZIP)

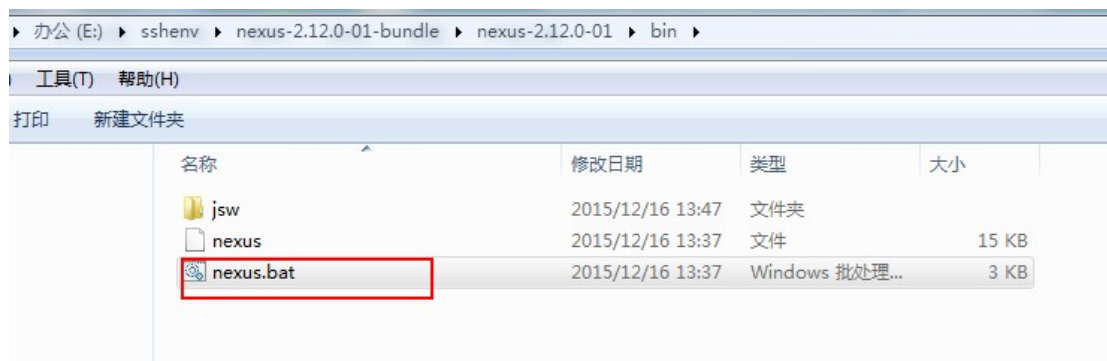
Checksums: MD5 SHA Signature: PGP

下载：nexus-2.12.0-01-bundle.zip

传智黑马课程 ▶ 黑马28期 ▶ maven ▶ day02 ▶ 资料 ▶ nexus私服 ▶				
功(H)				
名称	修改日期	类型	大小	
 nexus-2.12.0-01-bundle.tar.gz	2016/3/12 14:22	WinRAR 压缩文件	70,285 KB	
 nexus-2.12.0-01-bundle.zip	2016/3/12 14:06	WinRAR ZIP 压缩...	70,388 KB	

安装：

1.解压，进入指定的目录

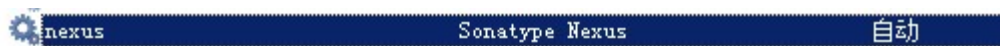


2.安装并启动这个应用程序

cmd 进入 bin 目录，执行 nexus.bat install (尽量采用管理员模式)

```
F:\nexus-2.12.0-01-bundle\nexus-2.12.0-01\bin>nexus.bat install
wrapper ! nexus installed.
```

安装成功在服务中查看有 nexus 服务：



通过从 cmd 端启动后如果出现问题提示，'findstr' 不是内部或外部命令，也不是可运行的程序或批处理文件；

这是 PATH 环境变量的问题，将 windows 命令的目录添加到 PATH 中就好了。

即：在 path 中追加：%SystemRoot%/system32;%SystemRoot%;**这两个**

卸载 nexus

cmd 进入 nexus 的 bin 目录，执行：nexus.bat uninstall

```
F:\nexus-2.12.0-01-bundle\nexus-2.12.0-01\bin>nexus.bat uninstall
```

查看 window 服务列表 nexus 已被删除。

启动 nexus

方法 1：

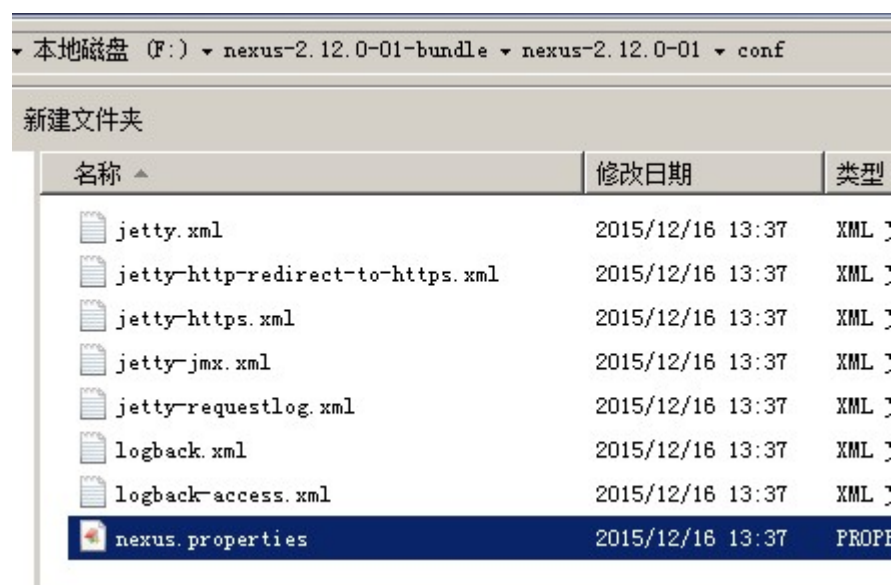
cmd 进入 bin 目录，执行 nexus.bat start

方法 2：

直接启动 nexus 服务



查看 nexus 的配置文件 conf/nexus.properties



Jetty section

application-port=8081 # nexus 的访问端口配置

application-host=0.0.0.0 # nexus 主机监听配置(不用修改)

nexus-webapp=\${bundleBasedir}/nexus # nexus 工程目录

nexus-webapp-context-path=/nexus # nexus 的 web 访问路径

Nexus section

nexus-work=\${bundleBasedir}/../sonatype-work/nexus # nexus 仓库目录

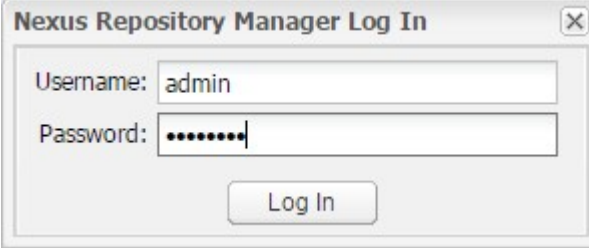
runtime=\${bundleBasedir}/nexus/WEB-INF # nexus 运行程序目录

访问：

<http://localhost:8081/nexus/>

使用 Nexus 内置账户 admin/admin123 登陆：

点击右上角的 Log in，输入账号和密码 登陆



A screenshot of the Nexus Repository Manager Log In dialog box. It has a title bar with the text "Nexus Repository Manager Log In" and a close button. Inside the dialog, there are two input fields: "Username:" with the text "admin" and "Password:" with masked characters ".....". Below the password field is a "Log In" button.

登陆成功：

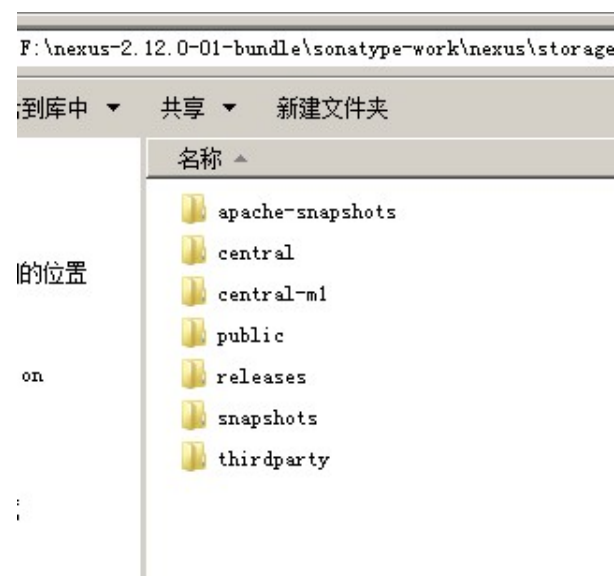


nexus 的仓库有 4 种类型：

Repository ▲	Type
Public Repositories	group
3rd party	hosted
Apache Snapshots	proxy
Central	proxy
Central M1 shadow	virtual
Releases	hosted
Snapshots	hosted

1. hosted，宿主仓库，部署自己的 jar 到这个类型的仓库，包括 releases 和 snapshot 两部分，Releases 公司内部发布版本仓库、 Snapshots 公司内部测试版本仓库
2. proxy，代理仓库，用于代理远程的公共仓库，如 maven 中央仓库，用户连接私服，私服自动去中央仓库下载 jar 包或者插件。
3. **group，仓库组，用来合并多个 hosted/proxy 仓库，通常我们配置自己的 maven 连接仓库组。**
4. virtual(虚拟)：兼容 Maven1 版本的 jar 或者插件

nexus 仓库默认在 sonatype-work 目录中：



✓ **central : 代理仓库，代理中央仓库**

The screenshot shows the Nexus Repository Manager configuration for a repository named 'central'. The 'Configuration' tab is selected. The following fields are visible:

- Repository ID: central
- Repository Name: Central
- Repository Type: proxy
- Provider: Maven2
- Format: maven2
- Repository Policy: Release (annotated as '发布版本')
- Default Local Storage Location: file:/F:/nexus-2.12.0-01-bundle/sonatype-work/nexus/storage/central/ (annotated as '仓库位置')
- Override Local Storage Location: (empty)
- Remote Repository Access: (expanded)
- Remote Storage Location: https://repo1.maven.org/maven2/ (annotated as '中央仓库地址')

✓ **apache-snapshots : 代理仓库**

存储 snapshots 构件，代理地址 <https://repository.apache.org/snapshots/>

- ✓ **central-m1** : virtual 类型仓库，兼容 Maven1 版本的 jar 或者插件
- ✓ **releases** : 本地仓库，存储 releases 构件。
- ✓ **snapshots** : 本地仓库，存储 snapshots 构件。
- ✓ **thirdparty** : 第三方仓库
- ✓ **public** : 仓库组

需求：将 ssh_dao 的这个工程打成 jar 包，并放入到私服上去。

配置

第一步：需要在客户端即部署 dao 工程的电脑上配置 maven 环境，并修改 settings.xml 文件，配置连接私服的用户和密码。

此用户名和密码用于私服校验，因为私服需要知道上传的账号和密码是否和私服中的账号和密码一致。

```
<server>

  <id>releases</id>

  <username>admin</username>

  <password>admin123</password>

</server>
```

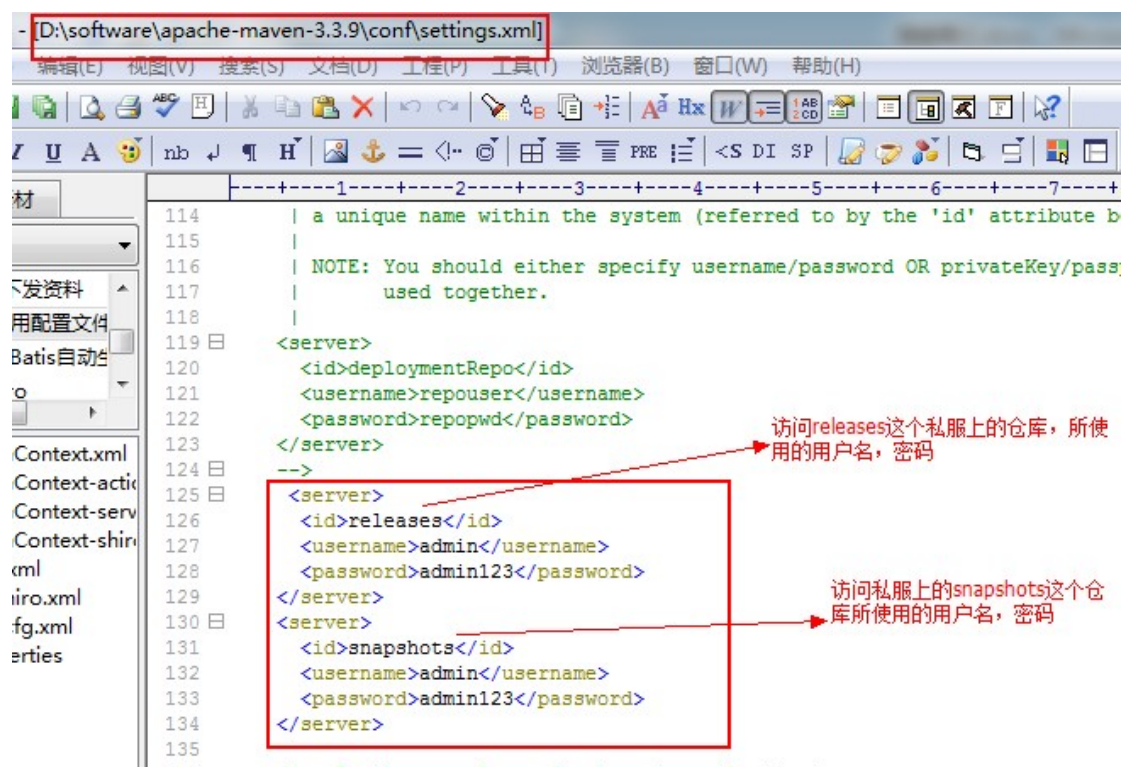
```
  <server>

    <id>snapshots</id>

    <username>admin</username>

    <password>admin123</password>

  </server>
```



releases 连接发布版本项目仓库

snapshots 连接测试版本项目仓库

Releases	hosted	ANALYZE	maven2	Release	In Service
Snapshots	hosted	ANALYZE	maven2	Snapshot	In Service

第二步：配置项目 pom.xml

配置私服仓库的地址，本公司的自己的 jar 包会上传到私服的宿主仓库，根据工程的版本号决定上传到哪个宿主仓库，如果版本为 release 则上传到私服的 release 仓库，如果版本为 snapshot 则上传到私服的 snapshot 仓库

```
<distributionManagement>
  <repository>
    <id>releases</id>
    <url>http://localhost:8081/nexus/content/repositories/releases/</url>
  </repository>
  <snapshotRepository>
    <id>snapshots</id>
    <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

注意：pom.xml 这里<id> 和 settings.xml 配置 <id> 对应！

测试

将项目 dao 工程打成 jar 包发布到私服：

- 1、首先启动 nexus
- 2、对 dao 工程执行 deploy 命令

从私服下载 jar 包

需求

没有配置 nexus 之前，如果本地仓库没有，去中央仓库下载，通常在企业中会在局域网内部署一台私服服务器，有了私服本地项目首先去本地仓库找 jar，如果没有

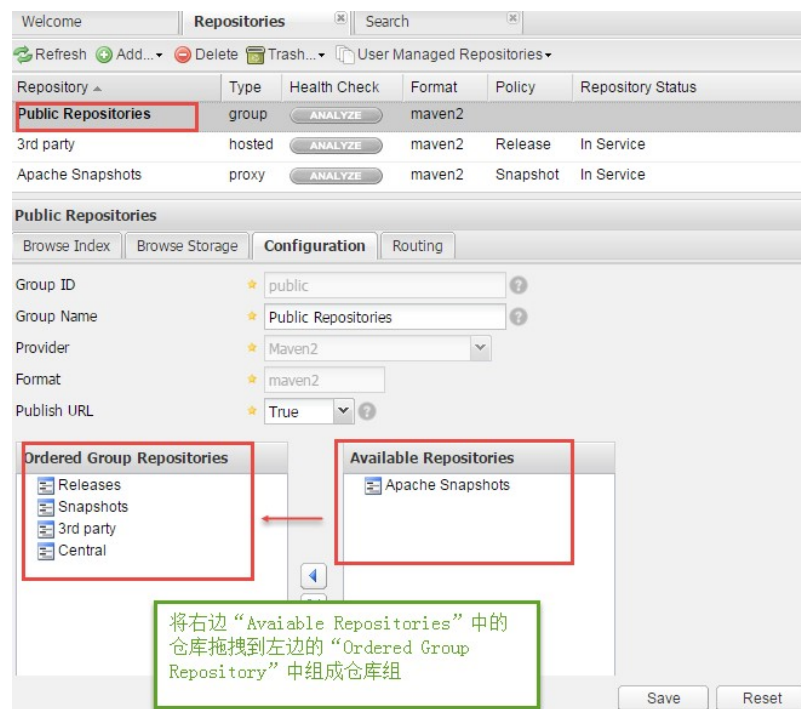
找到则连接私服从私服下载 jar 包，如果私服没有 jar 包私服同时作为代理服务器从中央仓库下载 jar 包，这样做的好处是一方面由私服对公司项目的依赖 jar 包统一管理，一方面提高下载速度，项目连接私服下载 jar 包的速度要比项目连接中央仓库的速度快的多。

本例子测试从私服下载 dao 工程 jar 包。

管理仓库组

nexus 中包括很多仓库，hosted 中存放的是企业自己发布的 jar 包及第三方公司的 jar 包，proxy 中存放的是中央仓库的 jar，为了方便从私服下载 jar 包可以将多个仓库组成一个仓库组，每个工程需要连接私服的仓库组下载 jar 包。

打开 nexus 配置仓库组，如下图：



上图中仓库组包括了本地仓库、代理仓库等。

在 setting.xml 中配置仓库

在客户端的 setting.xml 中配置私服的仓库，由于 setting.xml 中没有 repositories 的配置标签需要使用 profile 定义仓库。

```
<profile>

    <!--profile 的 id-->
    <id>dev</id>
    <repositories>
        <repository>

            <!--仓库 id，repositories 可以配置多个仓库，保证 id 不重复-->
            <id>nexus</id>

            <!--仓库地址，即 nexus 仓库组的地址-->
            <url>http://localhost:8081/nexus/content/groups/public/</url>

            <!--是否下载 releases 构件-->
            <releases>
                <enabled>true</enabled>
            </releases>

            <!--是否下载 snapshots 构件-->
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
    </repositories>

    <pluginRepositories>

        <!-- 插件仓库，maven 的运行依赖插件，也需要从私服下载插件 -->
        <pluginRepository>

            <!-- 插件仓库的 id 不允许重复，如果重复后边配置会覆盖前边 -->
            <id>public</id>

            <name>Public Repositories</name>

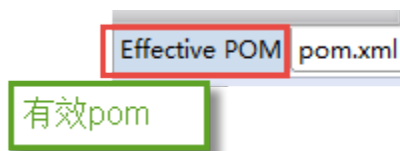
            <url>http://localhost:8081/nexus/content/groups/public/</url>
```

```
</pluginRepository>
</pluginRepositories>
</profile>
```

使用 profile 定义仓库需要激活才可生效。

```
<activeProfiles>
  <activeProfile>dev</activeProfile>
</activeProfiles>
```

配置成功后通过 eclipse 查看有效 pom，有效 pom 是 maven 软件最终使用的 pom 内容，程序员不直接编辑有效 pom，打开有效 pom



有效 pom 内容如下：

下边的 pom 内容中有两个仓库地址，maven 会先从前边的仓库的找，如果找不到 jar 包再从下边的找，从而就实现了从私服下载 jar 包。

```
<repositories>
  <repository>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
    <id>public</id>
    <name>Public Repositories</name>
    <url>http://localhost:8081/nexus/content/groups/public/</url>
  </repository>
  <repository>
    <snapshots>
```



```
        <enabled>false</enabled>

    </snapshots>

    <id>central</id>

    <name>Central Repository</name>

    <url>https://repo.maven.apache.org/maven2</url>

</repository>

</repositories>

<pluginRepositories>

    <pluginRepository>

        <id>public</id>

        <name>Public Repositories</name>

        <url>http://localhost:8081/nexus/content/groups/public/</url>

    </pluginRepository>

    <pluginRepository>

        <releases>

            <updatePolicy>never</updatePolicy>

        </releases>

        <snapshots>

            <enabled>false</enabled>

        </snapshots>

        <id>central</id>

        <name>Central Repository</name>

        <url>https://repo.maven.apache.org/maven2</url>

    </pluginRepository>

</pluginRepositories>
```