

**项目管理工具**

**maven**

# 教学计划

|  |  |  |  |
| --- | --- | --- | --- |
| 课时 | 章节 | 知识点 | 备注 |
| 一 | 传递依赖  依赖版本冲突解决  SSH构建 | 传递依赖 | 了解 |
| 依赖版本冲突解决 | 重点 |
| SSH构建 | 重点 |
| 二 | SSH构建  分模块构建SSH工程 | SSH构建 | 重点 |
| 分模块构建工程分析 | 理解 |
| 分模块构建工程-父工程 | 重点 |
| 分模块构建工程-Dao工程 | 重点 |
| 分模块构建工程-Service工程 | 重点 |
| 三 | 分模块构建SSH工程  Maven私服 | 分模块构建工程-web工程 | 重点 |
| 分模块构建工程-调试 | 重点 |
| 私服介绍、安装配置 | 了解 |
|  |  |
| 四 | Maven私服 | 私服-将项目发布到私服 | 了解 |
| 私服-从私服下载jar | 了解 |
| 知识点总结 |  |
|  |  |

思路：

在第一天的dao、service、web工程的基础上补充，

dao：集成hibernate和spring

service：集成spring的事务管理

web集成struts2+spring

调试通过

注意：

目前三个工程是是互相的依赖，但是只运行了web，如果dao或service修改，及时install到仓库！！！

等到后边的聚合运行，如果dao或service修改就不用install到本地仓库，直接聚合运行！！

# 依赖管理-传递依赖

## 传递依赖

### 什么是传递依赖

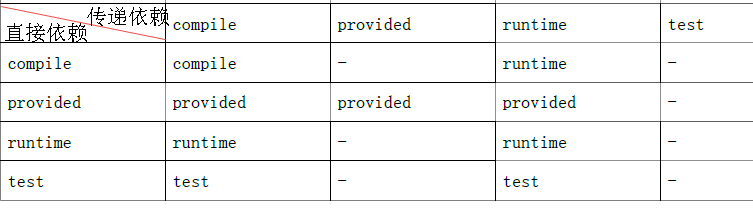
当A 依赖B、B依赖C，在A中导入B后会自动导入C，C是A的传递依赖，如果C依赖D则D也可能是A的传递依赖。

演示：

web中添加struts-spring的jar，传递依赖了spring

### 依赖范围对传递依赖的影响（了解）

依赖会有依赖范围，依赖范围对传递依赖也有影响，有A、B、C，A依赖B、B依赖C，C可能是A的传递依赖，如下图：



最左边一列为直接依赖，理解为A依赖B的范围，最顶层一行为传递依赖，理解为B依赖C的范围，行与列的交叉即为A传递依赖C的范围。

举例：

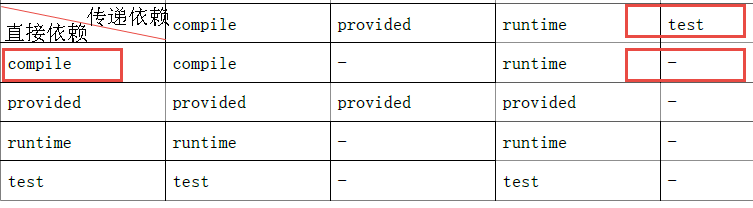
比如 A对 B 有 compile 依赖，B 对C有 runtime 依赖，那么根据表格所示A对C 有 runtime 依赖。

**测试**

dao依赖junit，scop为test

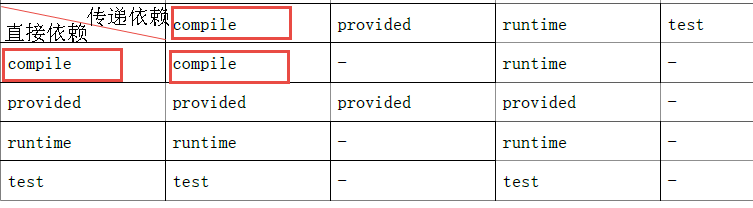
service依赖dao.

查看下图红色框内所示传递依赖范围：



所以maven-first所依赖的junit的jar没有加入到maven-web工程。

如果修改maven-first依赖junit的scop为compile，maven-first所依赖的junit的jar包会加入到maven-web工程中，符合上边表格所示，查看下图红色框内所示：



## 依赖版本冲突解决

### 问题

当一个项目依赖的构件比较多时，它们相互之前存在依赖，当你需要对依赖版本统一管理时如果让maven自动来处理可能并不能如你所愿，如下例子：

同时加入以下依赖，观察依赖：

<!-- struts2-spring-plugin依赖spirng-beans-3.0.5 -->

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>2.3.24</version>

</dependency>

<!-- spring-context依赖spring-beans-4.2.4 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

org.apache.struts依赖spirng-beans-3.0.5，spring-context依赖spring-beans-4.2.4，但是发现spirng-beans-3.0.5加入到工程中，而我们希望spring-beans-4.2.4加入工程。

### 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 spirng-beans-4.2.4，A依赖B依赖 spirng-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spirng-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spirng-beans-4.2.4的依赖，根据路径近者优先原则，系统将导入spirng-beans-4.2.4：

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

### 排除依赖

上边的问题也可以通过排除依赖方法辅助依赖调解，如下：

比如在依赖struts2-spring-plugin的设置中添加排除依赖，排除spring-beans，

下边的配置表示：依赖struts2-spring-plugin，但排除struts2-spring-plugin所依赖的spring-beans。

<!-- struts2-spring-plugin依赖spirng-beans-3.0.5 -->

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>2.3.24</version>

<!-- 排除 spring-beans-->

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

</exclusion>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</exclusion>

</exclusions>

</dependency>

### 锁定版本

面对众多的依赖，有一种方法不用考虑依赖路径、声明优化等因素可以采用直接锁定版本的方法确定依赖构件的版本，版本锁定后则不考虑依赖的声明顺序或依赖的路径，以锁定的版本的为准添加到工程中，此方法在企业开发中常用。

如下的配置是锁定了spring-beans和spring-context的版本：

<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>

注意：在工程中锁定依赖的版本并不代表在工程中添加了依赖，如果工程需要添加锁定版本的依赖则需要单独添加<dependencies></dependencies>标签，如下：

<dependencies>

<!--这里是添加依赖 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

</dependencies>

上边添加的依赖并没有指定版本，原因是已在<dependencyManagement>中锁定了版本，所以在<dependency>下不需要再指定版本。

# maven构建ssh工程

## 需求

在web工程的基础上实现ssh工程构建，规范依赖管理。

## 数据库环境

创建数据库：maven

导入sql/cst\_customer.sql创建表



## 定义pom.xml

maven工程首先要识别依赖，web工程实现SSH整合，需要依赖struts2.3.24、 spring4.2.4、hibernate5.0.7等，在pom.xml添加工程如下依赖：

分两步：

1）锁定依赖版本

2）添加依赖

<project xmlns=*"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>cn.itcast.maven</groupId>

<artifactId>maven-web-0120</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>web工程，包括jsp、action等</name>

<description>web工程，包括jsp、action等</description>

<!-- 属性 -->

<properties>

<spring.version>4.2.4.RELEASE</spring.version>

<hibernate.version>5.0.7.Final</hibernate.version>

<struts.version>2.3.24</struts.version>

</properties>

<!-- 锁定版本，struts2-2.3.24、spring4.2.4、hibernate5.0.7 -->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>${hibernate.version}</version>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-core</artifactId>

<version>${struts.version}</version>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>${struts.version}</version>

</dependency>

</dependencies>

</dependencyManagement>

<!-- 依赖管理 -->

<dependencies>

<!-- spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<!-- hibernate -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

</dependency>

<!-- 数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

<scope>runtime</scope>

</dependency>

<!-- c3p0 -->

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

<!-- 导入 struts2 -->

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-core</artifactId>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

</dependency>

<!-- servlet jsp -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.2</version>

</dependency>

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

</dependency>

<!-- jstl -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 设置编译版本为1.7 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<!-- maven内置 的tomcat6插件 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>tomcat-maven-plugin</artifactId>

<version>1.1</version>

<configuration>

<!-- 可以灵活配置工程路径 -->

<path>/ssh</path>

<!-- 可以灵活配置端口号 -->

<port>8080</port>

</configuration>

</plugin>

</plugins>

</build>

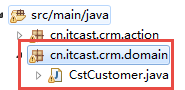
</project>

## dao

在src/main/java中定义dao接口，实现根据id 查询客户信息：

### domain模型类

在src/main/java创建模型类



**public** **class** CstCustomer **implements** java.io.Serializable {

**private** Long custId;

**private** String custName;

**private** Long custUserId;

**private** Long custCreateId;

**private** String custIndustry;

**private** String custLevel;

**private** String custLinkman;

**private** String custPhone;

**private** String custMobile;

...

### dao代码

**public** **class** CustomerDaoImpl **extends** HibernateDaoSupport **implements** CustomerDao {

@Override

**public** CstCustomer findCustomerById(Long custId) {

**return** **this**.getHibernateTemplate().get(CstCustomer.**class**, custId);

}

}

### 配置文件

* 在 src/main/resources创建hibernate.cfg.xml

<?xml version=*'1.0'* encoding=*'utf-8'*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<!-- 会话工厂 -->

<session-factory>

<!-- 数据库方言，根据数据库选择 -->

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5Dialect</property>

<!--为了方便调试是否在运行hibernate时在日志中输出sql语句 -->

<property name=*"hibernate.show\_sql"*>true</property>

<!-- 是否对日志中输出的sql语句进行格式化 -->

<property name=*"hibernate.format\_sql"*>true</property>

<property name=*"hibernate.hbm2ddl.auto"*>none</property>

<!-- 加载映射文件 -->

<mapping resource=*"cn/itcast/crm/domain/CstCustomer.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

* 在 src/main/resources创建applicationContext.xml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd"*>

<!-- 数据库连接池 -->

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/maven"* />

<property name=*"user"* value=*"root"* />

<property name=*"password"* value=*"mysql"* />

</bean>

<!-- 配置sessionFactory -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<!-- 依赖dataSource -->

<property name=*"dataSource"* ref=*"dataSource"*/>

<!-- 创建工厂需要加载hibernate映射文件 -->

<property name=*"configLocations"* value=*"classpath:hibernate.cfg.xml"*></property>

</bean>

<!-- dao -->

<bean id=*"customerDao"* class=*"cn.itcast.crm.dao.impl.CustomerDaoImpl"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"*/>

</bean>

</beans>

* 在src/main/resources配置log4j.properties

### direct log messages to stdout ###

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###

#在开发阶段日志级别使用debug

log4j.rootLogger=debug, stdout

### 在日志中输出sql的输入参数 ###

log4j.logger.org.hibernate.type=TRACE

### 单元测试

在src/test/java创建单元测试类

**public** **class** CustomerDaoImplTest {

@Test

**public** **void** testFindCustomerById() {

//获取spring容器

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

//获取dao

CustomerDao customerDao = (CustomerDao) applicationContext.getBean("customerDao");

//调用dao方法

CstCustomer cstCustomer = customerDao.findCustomerById(28l);

System.***out***.println(cstCustomer);

}

}

## service

### 代码

**public** **class** CustomerServiceImpl **implements** CustomerService {

// 定义dao

**private** CustomerDao customerDao;

@Override

**public** CstCustomer findCustomerById(Long custId) {

**return** customerDao.findCustomerById(custId);

}

}

### 配置文件

在applicationContext.xml中配置service

<!-- service -->

<bean id=*"customerService"* class=*"cn.itcast.crm.service.impl.CustomerServiceImpl"*>

<property name=*"customerDao"* ref=*"customerDao"*/>

</bean>

## action

### 代码

**public** **class** CustomerAction **extends** ActionSupport {

//依赖注入

**private** CustomerService customerService;

//客户id

**private** Long custId;

//客户信息

**private** CstCustomer customer;

//根据主键获取客户信息

**public** String querycustomer(){

customer = customerService.findCustomerById(custId);

**return** ***SUCCESS***;

}

**public** Long getCustId() {

**return** custId;

}

**public** **void** setCustId(Long custId) {

**this**.custId = custId;

}

**public** CstCustomer getCustomer() {

**return** customer;

}

**public** **void** setCustomer(CstCustomer customer) {

**this**.customer = customer;

}

}

### 配置文件

* 在applicationContext.xml中配置action

<!-- action -->

<bean id=*"customerAction"* class=*"cn.itcast.crm.action.CustomerAction"* scope=*"prototype"*>

<property name=*"customerService"* ref=*"customerService"*/>

</bean>

* 在struts.xml中配置action

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 配置常量 -->

<!-- 字符集 -->

<constant name=*"struts.i18n.encoding"* value=*"UTF-8"*></constant>

<!-- 开发模式 -->

<constant name=*"struts.devMode"* value=*"true"*></constant>

<!-- 主题 -->

<constant name=*"struts.ui.theme"* value=*"simple"*></constant>

<!-- 扩展名 -->

<constant name=*"struts.action.extension"* value=*"action"*></constant>

<!-- 通用package -->

<package name=*"customer"* namespace=*"/"* extends=*"struts-default"*>

<action name=*"querycustomer"* class=*"customerAction"*

method=*"querycustomer"*>

<result name=*"success"*>/jsp/querycustomer.jsp</result>

</action>

</package>

</struts>

* web.xml

加载spring容器，配置struts2前端控制器

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## Jsp

/src/main/webapp/jsp/querycustomer.jsp如下：

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>测试</title>

</head>

<body>

<!-- 从模型对象中获取属性值 -->

${customer.custName}

</body>

</html>

## 运行与调试

# 分模块构建工程

基于上边的三个工程分析

继承：创建一个parent工程将通用的pom配置抽取出来

聚合：聚合多个模块运行。

## 需求

### 需求描述

将ssh工程拆分为多个模块开发：

dao

service

web

parent父工程

ssh-web

ssh-service

ssh-dao

继承

继承

继承

依赖

依赖

### 理解继承和聚合

通常继承和聚合同时使用。

* 何为继承？

继承是为了消除重复，如果将dao、service、web分开创建独立的工程则每个工程的pom.xml文件中的内容存在重复，比如：设置编译版本、锁定spring的版本的等，可以将这些重复的配置提取出来在父工程的pom.xml中定义。

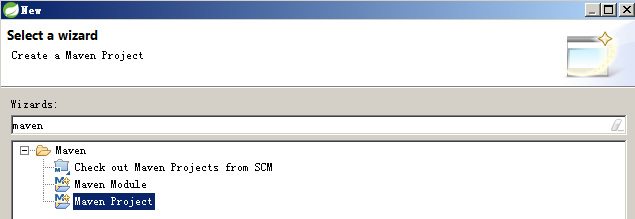
* 何为聚合？

项目开发通常是分组分模块开发，每个模块开发完成要运行整个工程需要将每个模块聚合在一起运行，比如：dao、service、web三个工程最终会打一个独立的war运行。

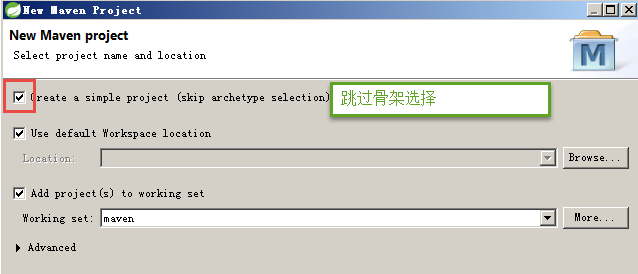
## 案例实现

### maven-parent父模块

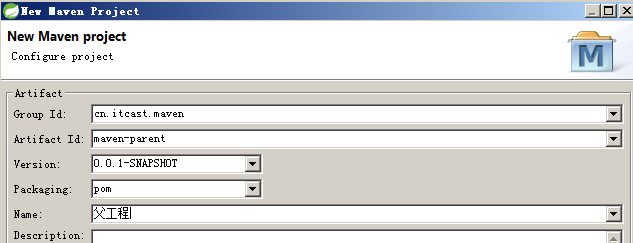
#### 创建父工程



这里选择“跳过骨架选择”



定义坐标：



#### 定义pom.xml

在父工程的pom.xml中抽取一些重复的配置的，比如：锁定jar包的版本、设置编译版本等。

<project xmlns=*"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>cn.itcast.maven</groupId>

<artifactId>maven-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<name>父工程</name>

<!-- 属性 -->

<properties>

<spring.version>4.2.4.RELEASE</spring.version>

<hibernate.version>5.0.7.Final</hibernate.version>

<struts.version>2.3.24</struts.version>

</properties>

<dependencyManagement>

<dependencies>

<!-- 统一依赖构件版本 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>${hibernate.version}</version>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-core</artifactId>

<version>${struts.version}</version>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>${struts.version}</version>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-json-plugin</artifactId>

<version>${struts.version}</version>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<finalName>maven-web</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

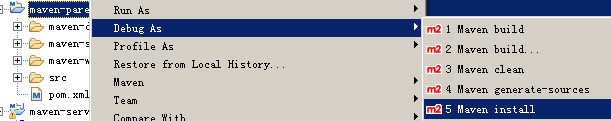
</plugins>

</build>

</project>

#### 将父工程发布至仓库

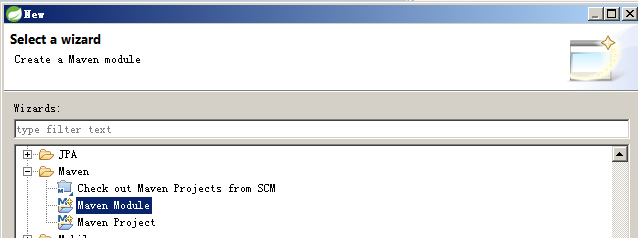
父工程创建完成执行maven-install将父工程发布到仓库方便子工程继承：



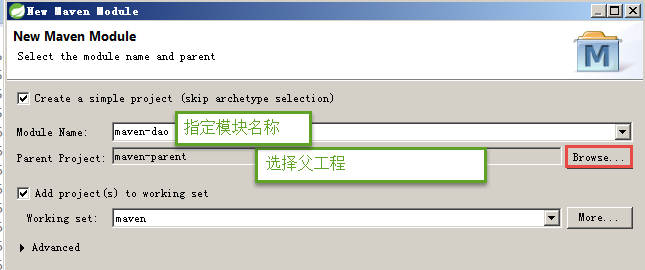
### maven-dao子模块

#### 创建dao子模块

选择maven模块：



这里指定模块名称，选择“跳过骨架选择”：



#### 定义pom.xml

dao模块的pom.xml文件中需要继承父模块，添加持久层需要的依赖坐标：

<project xmlns=*"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>

<parent>

<groupId>cn.itcast.maven</groupId>

<artifactId>maven-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>maven-dao</artifactId>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<!-- 数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

<scope>runtime</scope>

</dependency>

<!-- c3p0 -->

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.2</version>

</dependency>

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

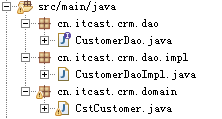
</dependency>

</dependencies>

</project>

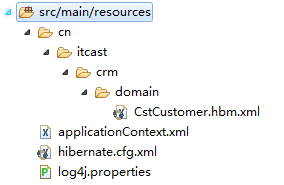
#### dao接口

将maven-web工程中的dao接口及domain类拷贝到src/main/java中：



#### 配置文件

拷贝maven-web工程中如下配置文件到dao工程：



注意：

将applicationContext.xml拆分出一个applicationContext-dao.xml，此文件中只配置dao

<!-- dao -->

<bean id=*"customerDao"* class=*"cn.itcast.crm.dao.impl.CustomerDaoImpl"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"*/>

</bean>

applicationContext.xml的内容如下：

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd"*>

<!-- 数据库连接池 -->

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/maven"* />

<property name=*"user"* value=*"root"* />

<property name=*"password"* value=*"mysql"* />

</bean>

<!-- 配置sessionFactory -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<!-- 依赖dataSource -->

<property name=*"dataSource"* ref=*"dataSource"*/>

<!-- 创建工厂需要加载hibernate映射文件 -->

<property name=*"configLocations"* value=*"classpath:hibernate.cfg.xml"*></property>

</bean>

#### 单元测试

@ContextConfiguration(locations = { "classpath:applicationContext.xml",

classpath:applicationContext-\*.xml })

@RunWith(SpringJUnit4ClassRunner.**class**)

**public** **class** CustomerDaoImplTest {

@Autowired

**private** CustomerDao customerDao;

@Before

**public** **void** setUp() **throws** Exception {

}

@Test

**public** **void** testFindCustomerById() {

CstCustomer customer = customerDao.findCustomerById(1l);

System.***out***.println(customer);

}

}

### maven-service子模块

#### 创建service子模块

方法同maven-dao模块创建方法，模块名称为maven-service。

#### 定义pom.xml

service模块的pom.xml文件中需要继承父模块，service依赖dao模块：

<project xmlns=*"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>

<parent>

<groupId>cn.itcast.maven</groupId>

<artifactId>maven-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>maven-service</artifactId>

<packaging>jar</packaging>

<dependencies>

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

</dependency>

<!-- 依赖dao -->

<dependency>

<groupId>cn.itcast.maven</groupId>

<artifactId>maven-dao</artifactId>

<version>0.0.1-SNAPSHOT</version>

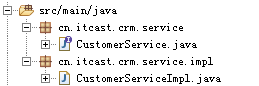
</dependency>

</dependencies>

</project>

#### service接口

将maven-web工程中的service接口拷贝到src/main/java中：



#### 配置文件

创建applicationContext-service.xml，此文件中定义的service。

<!-- service -->

<bean id=*"customerService"* class=*"cn.itcast.crm.service.impl.CustomerServiceImpl"*>

<property name=*"customerDao"* ref=*"customerDao"*/>

</bean>

#### 单元测试

@ContextConfiguration(locations = { "classpath:applicationContext.xml",

"classpath:applicationContext-\*.xml" })

@RunWith(SpringJUnit4ClassRunner.**class**)

**public** **class** CustomerServiceImplTest {

@Autowired

**private** CustomerService customerService;

@Before

**public** **void** setUp() **throws** Exception {

}

@Test

**public** **void** testFindCustomerById() {

CstCustomer customer = customerService.findCustomerById(1l);

System.***out***.println(customer);

}

}

### maven-web子模块

#### 创建web子模块

方法同maven-dao模块创建方法，模块名称为maven-web。

#### 定义pom.xml

web模块的pom.xml文件中需要继承父模块，web依赖service模块：

<project xmlns=*"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>

<parent>

<groupId>cn.itcast.maven</groupId>

<artifactId>maven-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>maven-web</artifactId>

<packaging>war</packaging>

<name>web工程</name>

<dependencies>

<!-- junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

</dependency>

<!-- 依赖service -->

<dependency>

<groupId>cn.itcast.maven</groupId>

<artifactId>maven-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

</dependency>

<!-- 导入 struts2 -->

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-core</artifactId>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

</dependency>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-json-plugin</artifactId>

</dependency>

<!-- servlet jsp -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

<!-- jstl -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

</project>

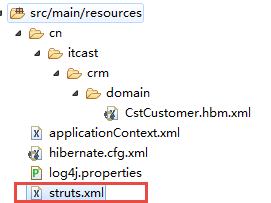
#### action

将maven-web工程中的action拷贝到src/main/java中：



#### 配置文件

拷贝maven-web工程中如下配置文件：



创建applicationContext-action.xml，此文件配置action

<!-- action -->

<bean id=*"customerAction"* class=*"cn.itcast.crm.action.CustomerAction"* scope=*"prototype"*>

<property name=*"customerService"* ref=*"customerService"*/>

</bean>

### 模块聚合

修改父工程的pom.xml，添加：

<!-- 由于父工程要聚合子工程，配置modules -->

<modules>

<!-- 在modules中配置相对路径，相对父工程pom.xml的路径找到子工程的pom.xml -->

<module>maven-dao</module>

<module>maven-service</module>

<module>maven-web</module>

</modules>

注意：上边module中配置的路径，此路径是相对父工程的pom.xml文件找子工程的pom.xml文件的路径。

### 运行调试

方法1：在maven-web工程的pom.xml中配置tomcat插件运行

运行maven-web工程它会从本地仓库下载依赖的jar包，所以当maven-web依赖的jar包内容修改了必须及时发布到本地仓库，比如：maven-web依赖的maven-service修改了，需要及时将maven-service发布到本地仓库。

方法2：在父工程的pom.xml中配置tomcat插件运行，自动聚合并执行

推荐方法2，如果子工程都在本地，采用方法2则不需要子工程修改就立即发布到本地仓库，父工程会自动聚合并使用最新代码执行。

注意：如果子工程和父工程中都配置了tomcat插件，运行的端口和路径以子工程为准。

# maven私服

正式开发，不同的项目组开发不同的工程。

dao工程开发完毕，发布到私服。

service从私服下载dao

## 需求

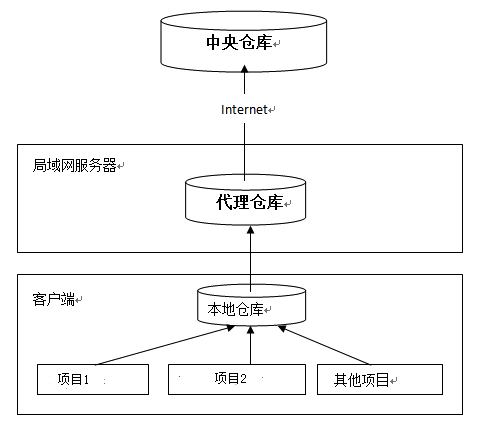
* 问题5：项目组编写了一个通用的工具类，其它项目组将类拷贝过去使用，当工具类修改bug后通过邮件发送给各各项目组，这种分发机制不规范可能导致工具类版本不统一。

解决方案：项目组将写的工具类通过maven构建，打成jar，将jar包发布到公司的maven仓库中，公司其它项目通过maven依赖管理从仓库自动下载jar包。

## 分析

公司在自己的局域网内搭建自己的远程仓库服务器，称为私服，私服服务器即是公司内部的maven远程仓库，每个员工的电脑上安装maven软件并且连接私服服务器，员工将自己开发的项目打成jar并发布到私服服务器，其它项目组从私服服务器下载所依赖的构件（jar）。

私服还充当一个代理服务器，当私服上没有jar包会从互联网中央仓库自动下载，如下图：



## 搭建私服环境

### 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。

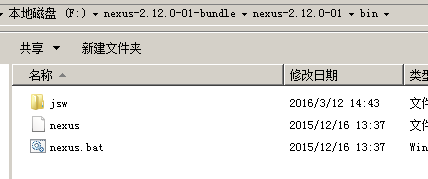
下载Nexus， 下载地址：http://www.sonatype.org/nexus/archived/



下载：nexus-2.12.0-01-bundle.zip

### 安装nexus

解压nexus-2.12.0-01-bundle.zip，本教程将它解压在F盘，进入bin目录：



cmd进入bin目录，执行nexus.bat install



安装成功在服务中查看有nexus服务：



### 卸载nexus

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



查看window服务列表nexus已被删除。

### 启动nexus

方法1：

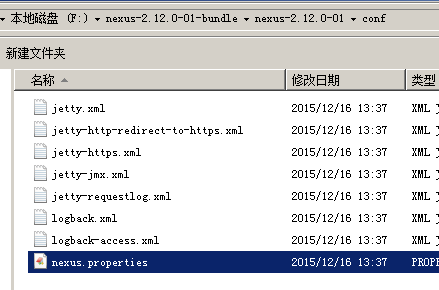
cmd进入bin目录，执行nexus.bat start

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

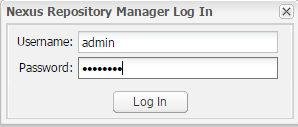
访问：

http://localhost:8081/nexus/

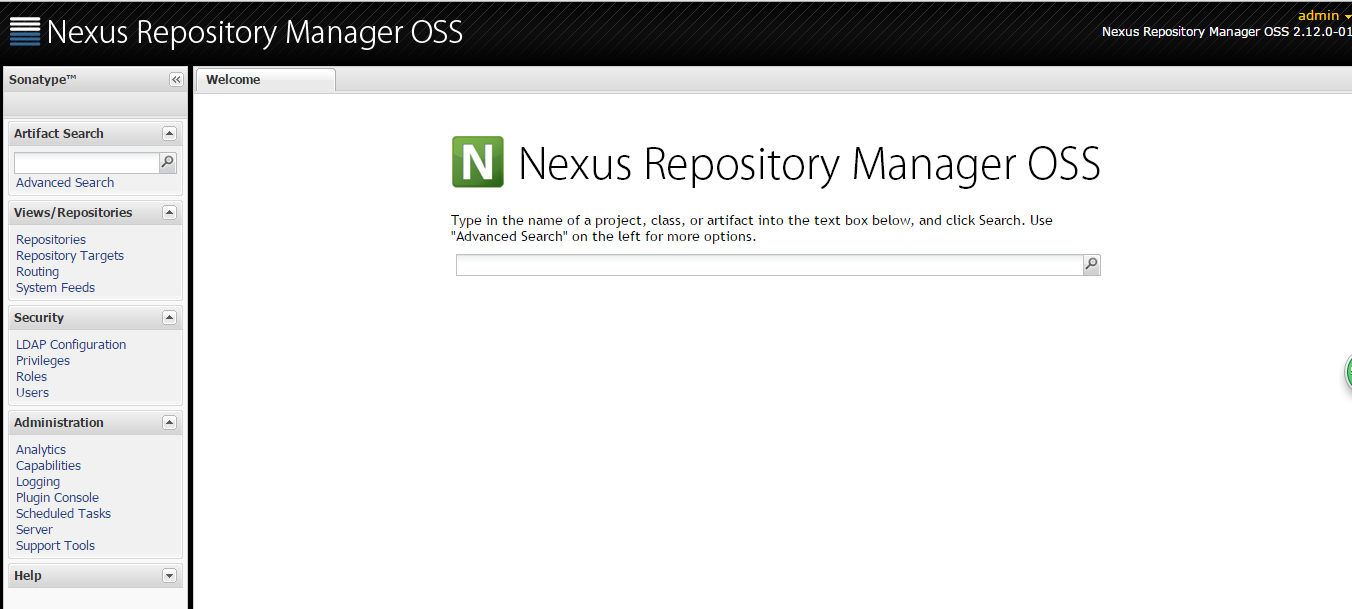


使用Nexus 内置账户admin/admin123登陆：

点击右上角的Log in，输入账号和密码 登陆



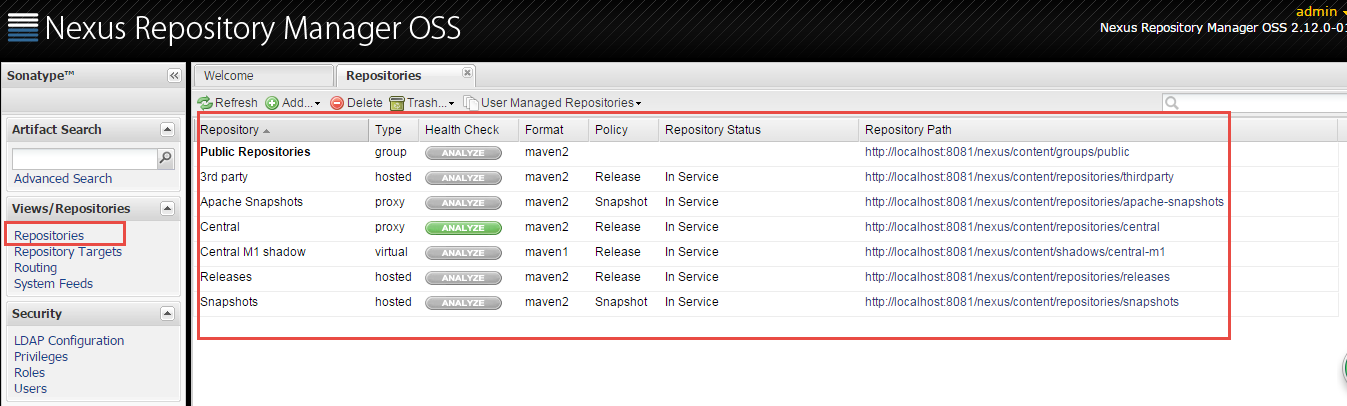
登陆成功：



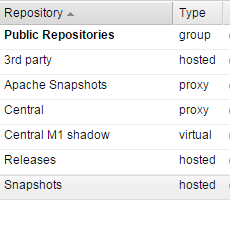
### 仓库类型

nexus

查看nexus的仓库：

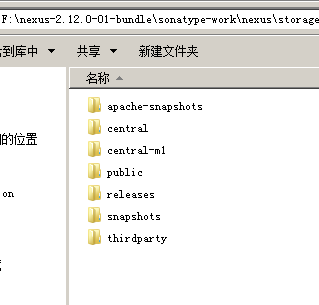


nexus的仓库有4种类型：



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**



* **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**

## 将项目发布到私服

### 需求

企业中多个团队协作开发通常会将一些公用的组件、开发模块等发布到私服供其它团队或模块开发人员使用。

本例子假设多团队分别开发dao、service、web，某个团队开发完在dao会将dao发布到私服供service团队使用，本例子会将dao工程打成jar包发布到私服。

私服(服务端)

nexus

dao工程

(客户端)

service工程

（客户端）

上传

上传

### 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都 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



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！

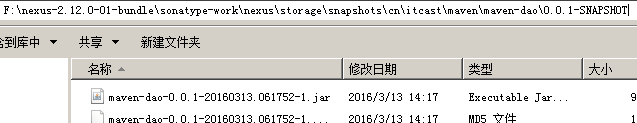
### 测试

将项目dao工程打成jar包发布到私服：

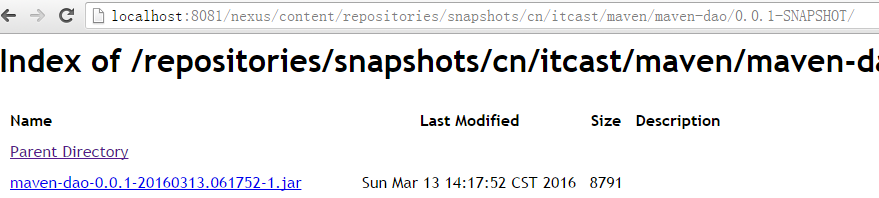
1、首先启动nexus

2、对dao工程执行deploy命令

根据本项目pom.xml中version定义决定发布到哪个仓库，如果version定义为snapshot，执行deploy后查看nexus的snapshot仓库，如果version定义为release则项目将发布到nexus的release仓库，本项目将发布到snapshot仓库：



也可以通过http方式查看：



## 从私服下载jar包

### 需求

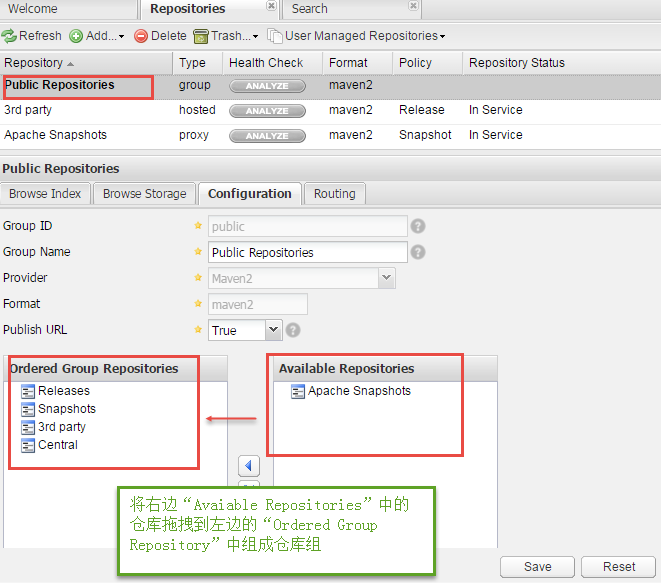
没有配置nexus之前，如果本地仓库没有，去中央仓库下载，通常在企业中会在局域网内部署一台私服服务器，有了私服本地项目首先去本地仓库找jar，如果没有找到则连接私服从私服下载jar包，如果私服没有jar包私服同时作为代理服务器从中央仓库下载jar包，这样做的好处是一方面由私服对公司项目的依赖jar包统一管理，一方面提高下载速度，项目连接私服下载jar包的速度要比项目连接中央仓库的速度快的多。

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>

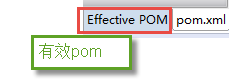
使用profile定义仓库需要激活才可生效。

<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>

### 测试从私服下载jar包

测试1：局域网环境或本地网络即可

在service工程中添加以上配置后，添加dao工程的依赖，删除本地仓库中dao工程，同时在eclipse中关闭dao工程。

观察控制台：

Downloading http://localhost:8081/nexus/content/groups/public/cn/itcast/ssh/ssh-dao0423/0.0.1-SNAPSHOT/ssh-dao0423-0.0.1-20160702.123702-1.jar

项目先从本地仓库找dao，找不到从私服找，由于之前执行deploy将dao部署到私服中，所以成功从私服下载dao并在本地仓库保存一份。

如果此时删除私服中的dao，执行update project之后是否正常？

如果将本地仓库的dao和私服的dao全部删除是否正常？

测试2：需要互联网环境

在项目的pom.xml添加一个依赖，此依赖在本地仓库和私服都不存在，maven会先从本地仓库找，本地仓库没有再从私服找，私服没有再去中央仓库下载，jar包下载成功在私服、本地仓库分别存储一份。

# 总结

## maven工程构建流程

1、创建maven工程，定义坐标

2、添加依赖

3、编写工程代码

4、如果是war工程使用tomcat插件运行

## maven工程分模块构建流程

1、创建父工程

父工程的pom.xml文件内容是将子工程中重复的pom内容进行抽取，通常包括锁定依赖版本、设置编译版本、tomat运行插件

2、创建各各子工程

3、聚合运行

在父工程中配置module聚合运行。

## 私服的应用

1、将自己的工程打包jar发布到私服

2、从私服下载jar包到自己的工程中