项目构建管理

# 一、SVN

## 1、基本介绍

SVN是目前非常流行版本管理工具

CVS

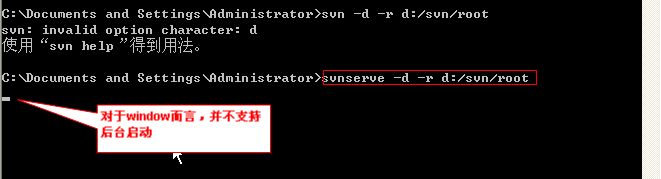
SubVersion

## 2、服务器

### 2.1、服务器的创建

1、创建一个根目录，这个根目录用来存储所有的svn的工厂信息（每一个工厂等于一个项目）

2、启动svn-->svnserve -d -r d:/svn/root



3、对于Linux而言，直接后台启动即可

对于window而言，后台启动无用，所以需要将其添加到window的服务中

graphic

### 2.2、开发流程

配置管理人员

1、创建工厂

graphic

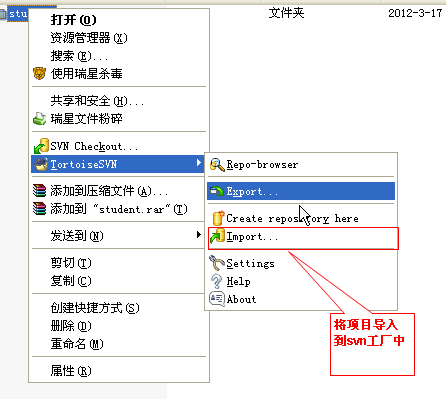
2、配置权限

3、由项目经理创建项目并且提交到svn工厂中

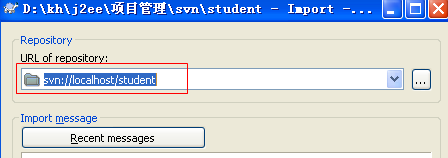
1、安装TortoiseSVN-1.7.6或者Eclipse中的插件

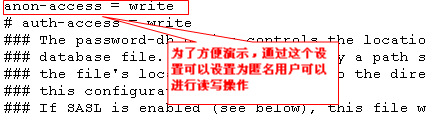


出现红色标志的按钮之后就说明安装成功！



4、





以上就正常的完成了导入

4、由开发人员进行checkout完成开发



## 3、客户端

### 3.1、安装

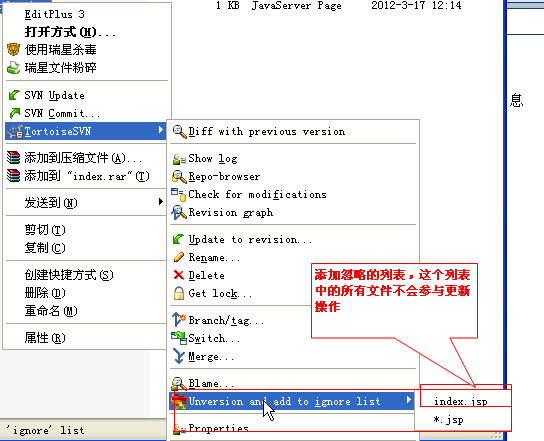
### 3.2、开发流程

1、checkout

2、开发完成之后-->commit

3、其他开发人员可以通过-->update更新svn仓库中的信息

4、



### 3.3、冲突问题

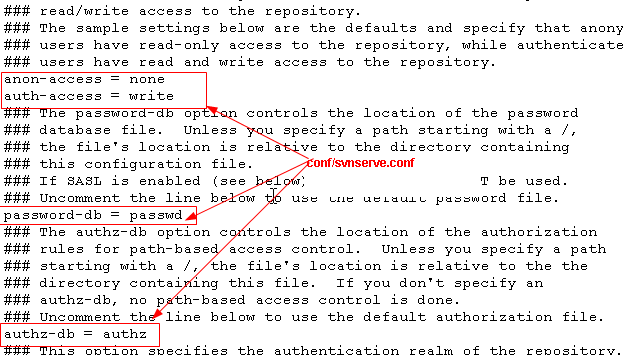
n两个开发人员共同修改了一个文件之后并且提交就会产生冲突

1、修改代码之前先进行更新

2、项目经理分配任务的时候需要独立的来考虑（为不同的开发人员分配不同的模块进行开发，这样尽可能的避免冲突）

### 3.4、权限控制

1、开启权限



**4、eclipse中的使用**

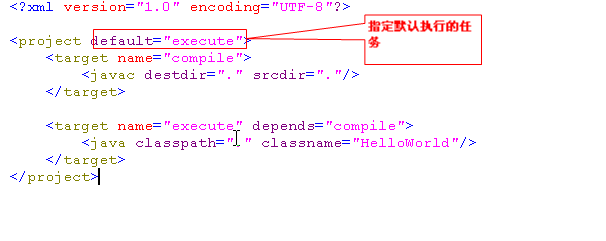
# 二、Ant

## 1、Ant的基础

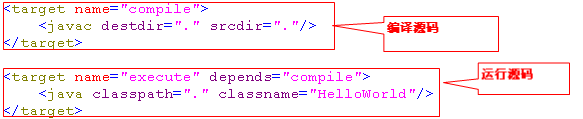
1.1、Ant的安装

1.2、Ant的Project

build.xml

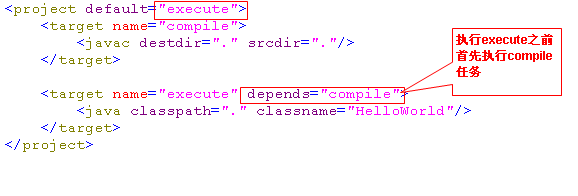
1.3、Ant的target

target表示的是每一项具体的任务

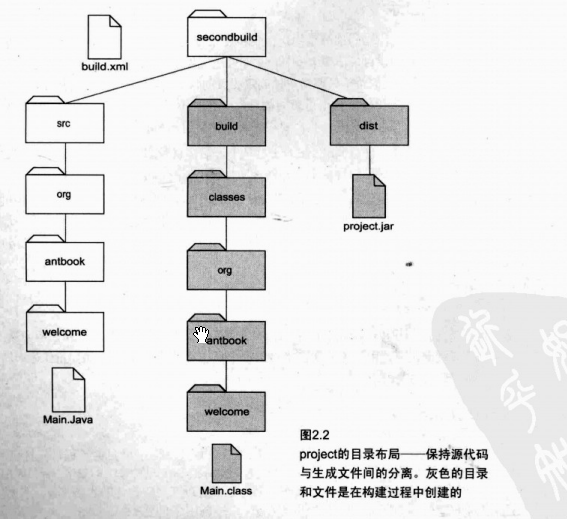


1.4、Ant的depends

当不同的任务之间有相应的关联的时候，可以通过depends属性来设定



1.5、最佳实践



1.6、ant编译普通java项目的流程

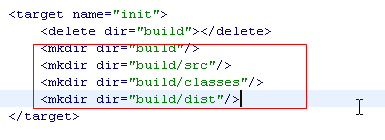
1、创建文件夹

build-->所有的文件信息都保存在这个文件夹中

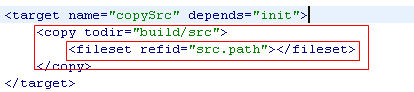
src-->所有的源码信息保存在这个文件夹中

classes-->编译好的所有文件保存在这个文件夹中

dist-->编译好的jar文件保存在这个文件夹中



2、将src文件夹的数据拷贝到bulid/src



3、编译源代码

graphic

4、将源代码打包为jar



5、直接运行程序



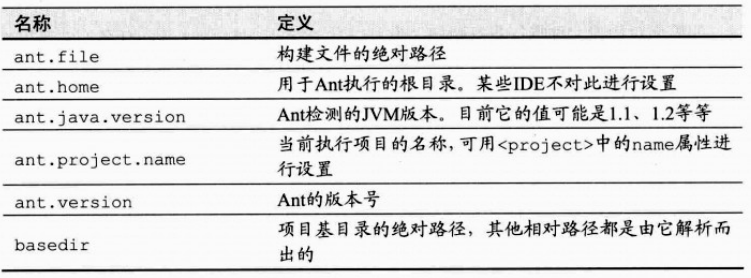
## 2、ant的文件集、路径和属性

2.1、ant的文件集



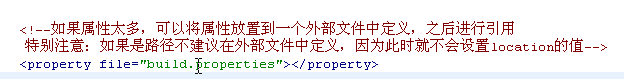
2.2、ant的文件路径

2.3、ant的属性



自定义属性





加载环境变量



2.4、ant的其他常用功能

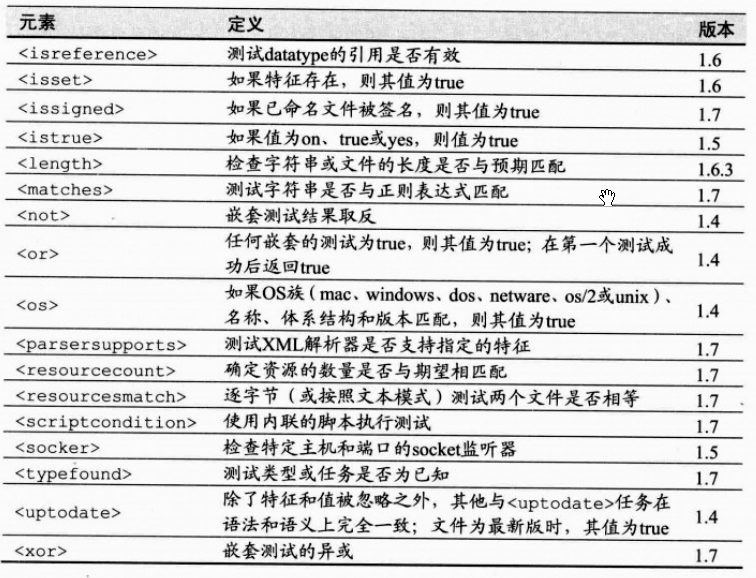
filter

available

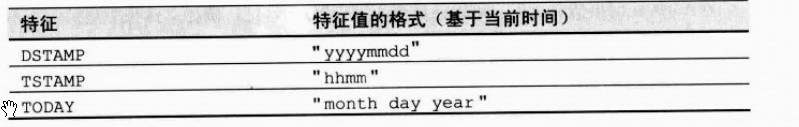
subtopic

condition





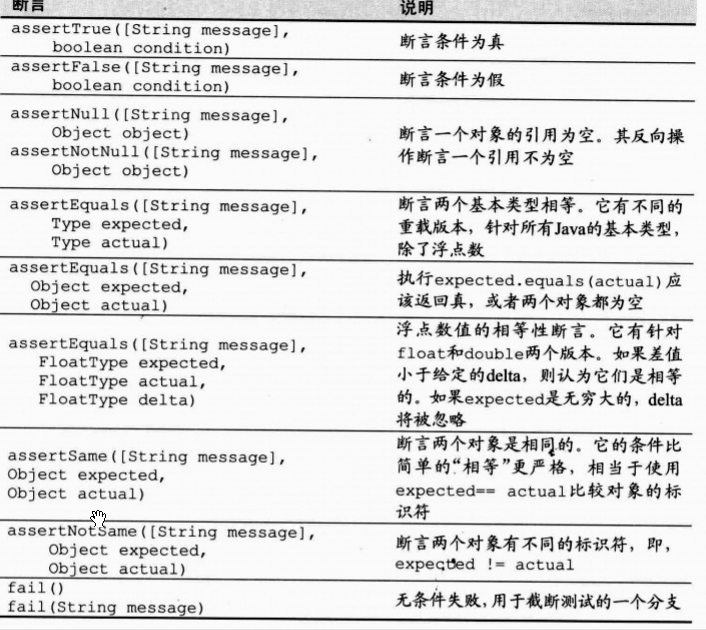
tstamp



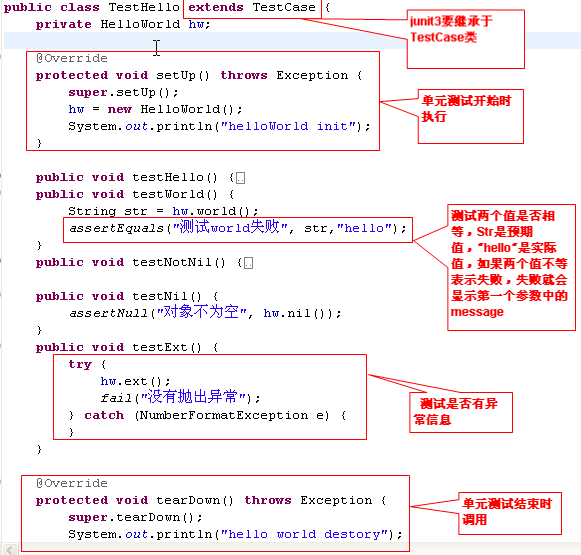
## 3、ant进行测试

3.1、JUnit

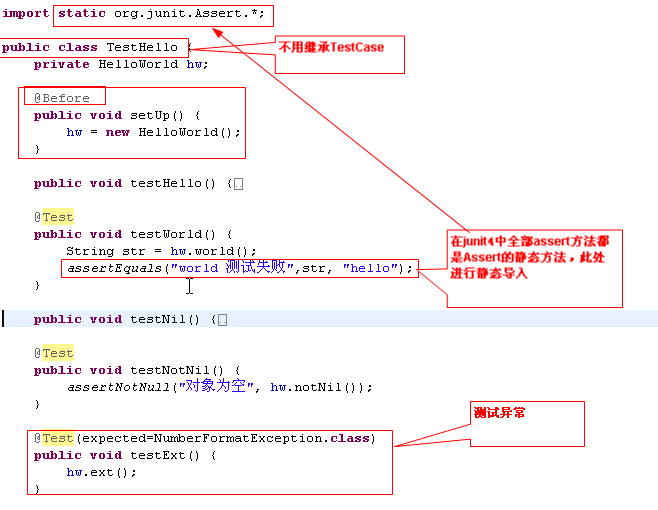
断言



基于3的版本

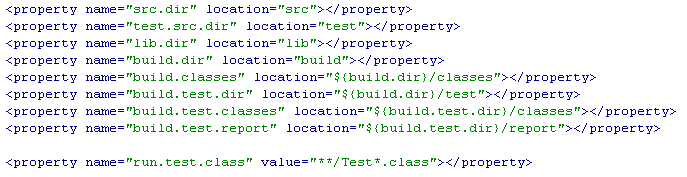


基于4 的版本



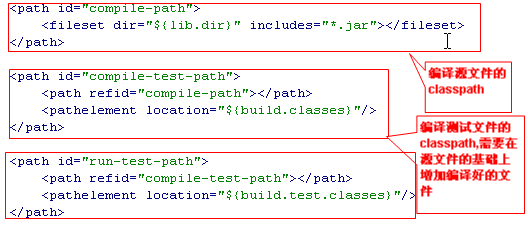
3.2、Ant进行测试的编译

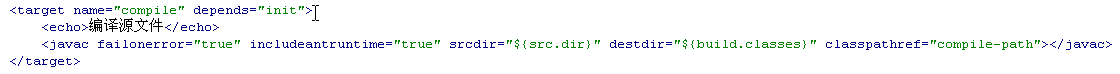
1、创建属性



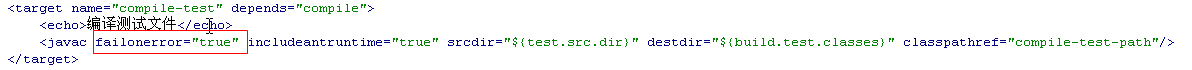
2、编译源文件（需要依赖于相应的classpath）

设置classpath路径

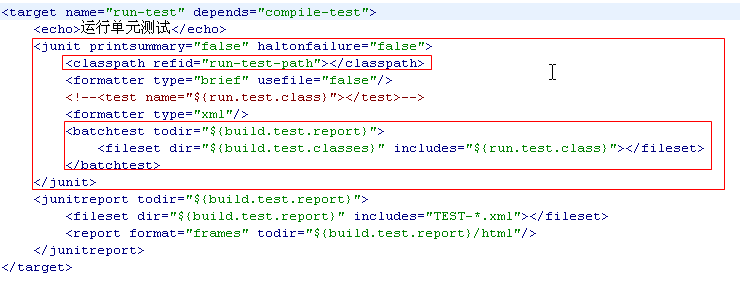




3、编译test的文件



4、运行单元测试



5、生成单元测试报告



3.3、生成报告



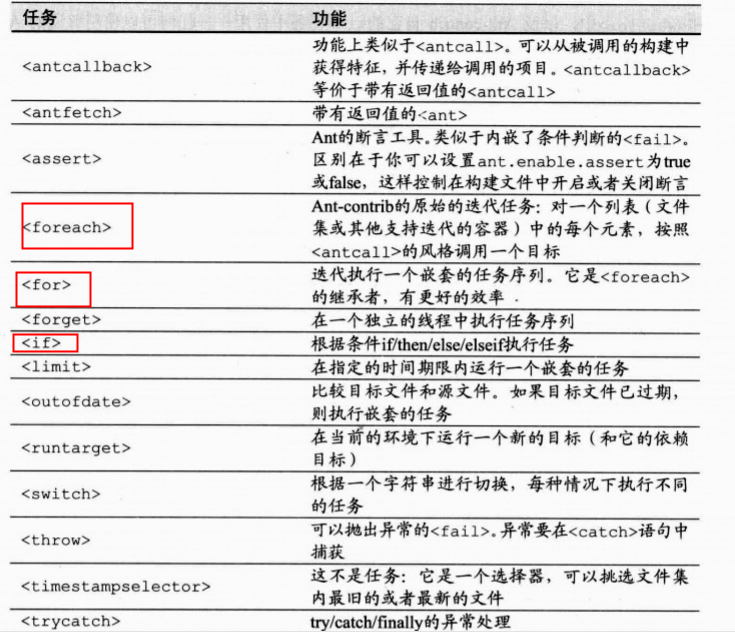
## 4、完成的项目实例(基于普通java程序)

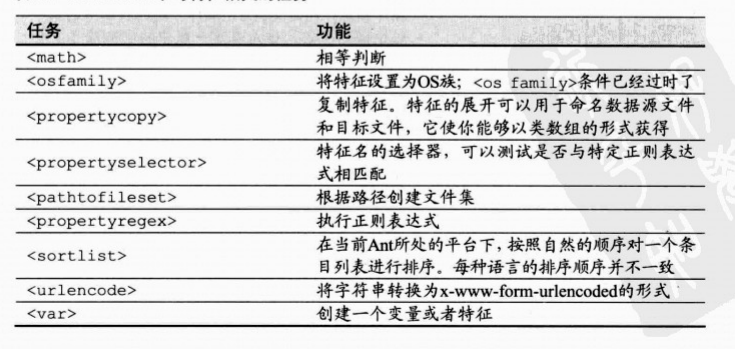


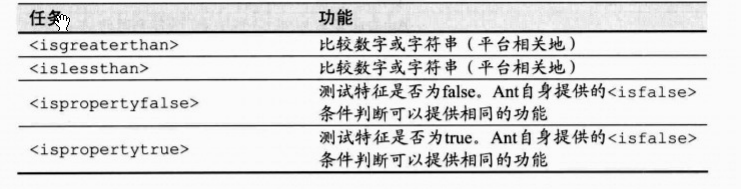
## 5、ant深入

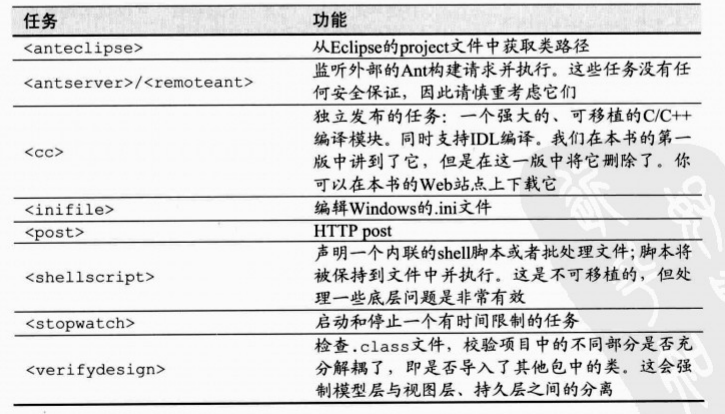
6.1、propfile

5.2、第三方任务



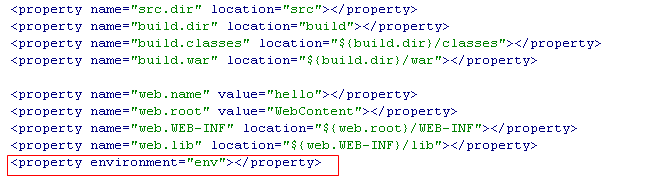




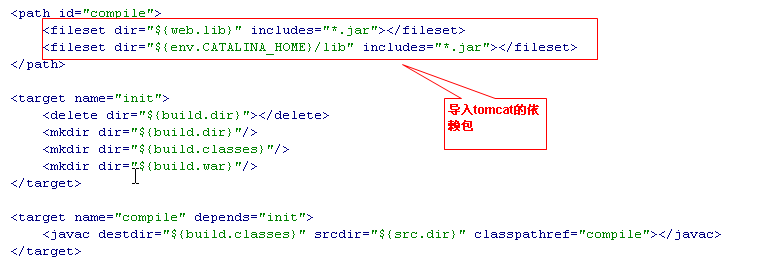


5.3、web发布

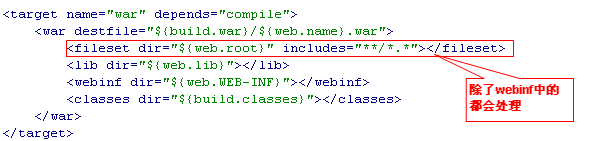
1、设置属性



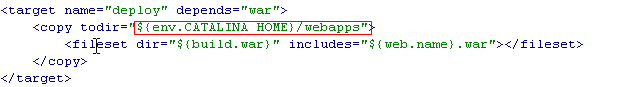
2、编译



3、打成war包



4、拷贝到tomcat/webapps



5.4、数据库设置



5.5、xml

**6、基于SpringMVC的用户管理项目**

# 三、Maven

## 1、maven的初步

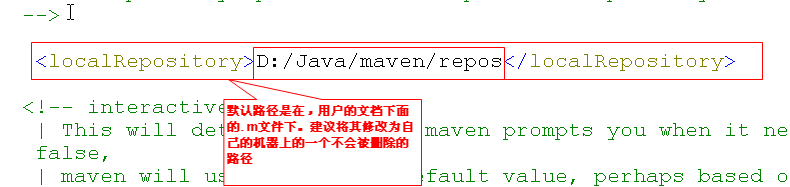
1.1、安装

下载maven的3.0.4版本

设置环境变量中的path

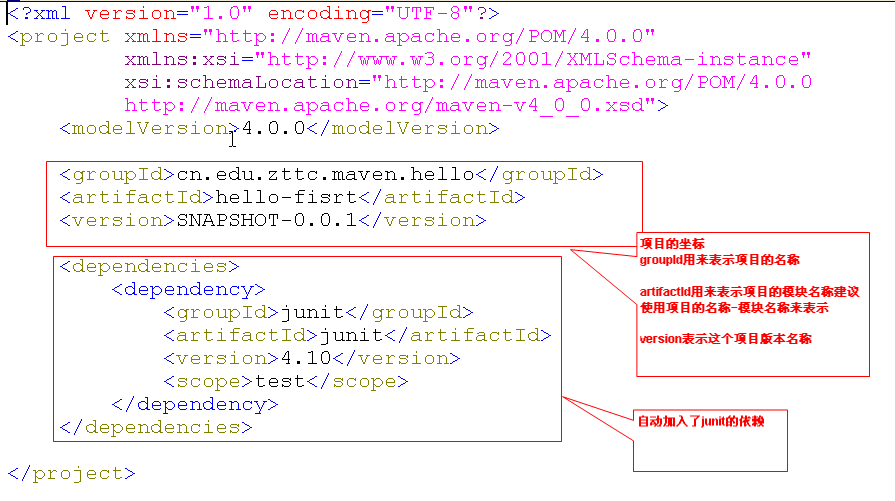
1.2、本地仓库的安装

1、修改maven目录下的conf中的setting.xml



1.3、新建一个简单的项目

主要创建文件pom.xml



1.4、创建代码和测试代码

源代码应该放置到src/main/java中

源代码的资源文件应该放置在src/main/resources文件夹中

测试代码应该放置到src/test/java中

测试代码的资源文件应该放置在src/test/resources文件夹中

1.5、运行mvn

mvn clean -->表示运行清理操作（会默认把target文件夹中的数据清理）

mvn clean compile-->表示先运行清理之后运行编译，会见代码编译到target文件夹中

mvn clean test-->运行清理和测试

mvn clean package-->运行清理和打包

mvn clean install-->运行清理和安装，会将打好的包安装到本地仓库中，以便其他的项目可以调用

mvn clean deploy-->运行清理和发布（发布到私服上面）

1.6、maven的生成方式

·由于maven有一些固定的生成模式，所以使用

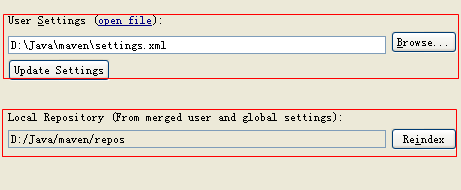
mvn archetype generate可以自动完成这个骨架的建立

·使用eclipse开发maven项目

默认STS和myeclipse都自带了maven的支持

如果是eclipse需要安装m2eclipse





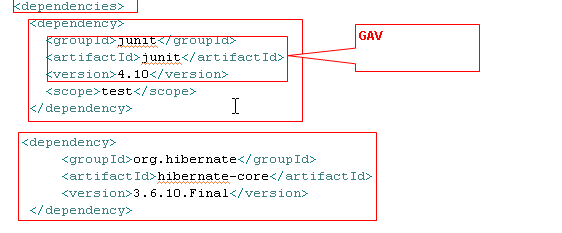
## 2、依赖

2.1、依赖包的查询

1、所有的依赖都是通过坐标来进行存储的（GAV-->groupId、artifactId、version）

2、有一些网上的仓库提供了坐标的查询（http://mvnrepository.com）

3、通过<dependencies>设置依赖

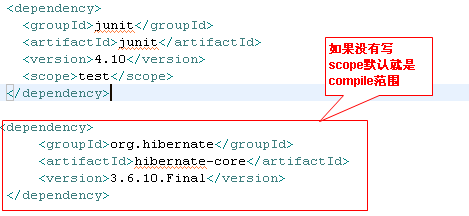


maven是如何搜索依赖的？首先会在本地仓库查询如果本地仓库没有，就去中央仓库查询

2.2、依赖的传递性

1、依赖是会被传递

A-->C B-->A ==> B-->C（这种依赖是基于compile这个范围进行传递）



对于依赖的传递而言，主要是针对compile作用域传递

传递的冲突问题

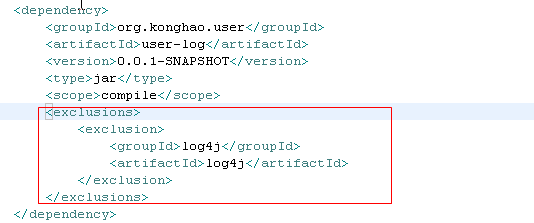
1、a-->b1.0 c-->b1.1 d-->a和c，这个时候在d的pom中，哪一个依赖先写就使用先写依赖的版本

如果d--><d>a</d><d>c</d>-->依赖版本是b1.0

2、a-->b1.0 c-->b1.1 d-->a和c-->b1.0 f-->d,c,如果路径的长短不一致就选择最小路径

f-->b1.1

3、如果希望精确的控制依赖包，可以使用依赖的排除功能进行控制



2.3、依赖的范围

1、test范围指的是测试范围有效，在编译和打包时都不会使用这个依赖

2、compile范围指的是编译范围有效，在编译和打包时都会将依赖存储进去

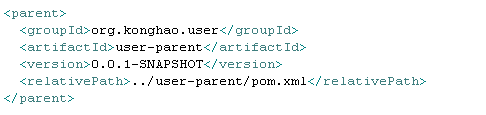
3、provided依赖：在编译和测试的过程有效，最后生成war包时不会加入，诸如：servlet-api，因为servlet-api，tomcat等web服务器已经存在了，如果再打包会冲突

4、runtime在运行的时候依赖，在编译的时候不依赖

默认的依赖范围是compile

2.4、聚合和继承





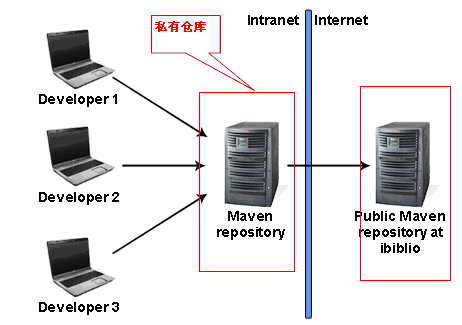
## 3、仓库

3.1、本地仓库

3.2、中心仓库

3.3、私有仓库

局域网内部的仓库



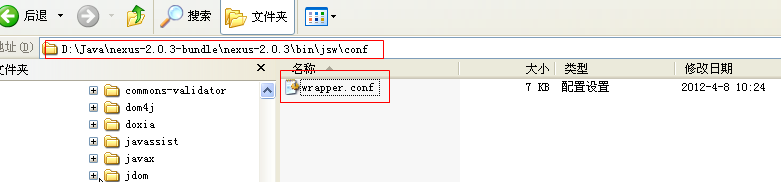
1、nexus的安装

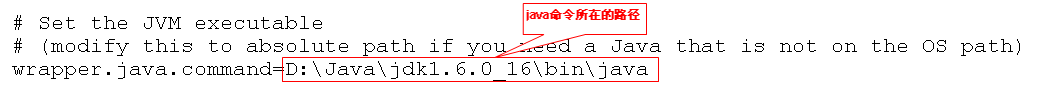
1、下载并且解压缩

2、将bin添加到环境变量

3、nexus install将nexus安装到windows的服务中

4、修改





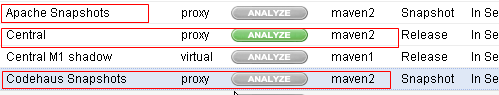
5、nexus start启动服务

2、仓库讲解

1、host的仓库。内部项目的发布仓库



2、proxy的仓库。从远程中央仓库中寻找数据的仓库



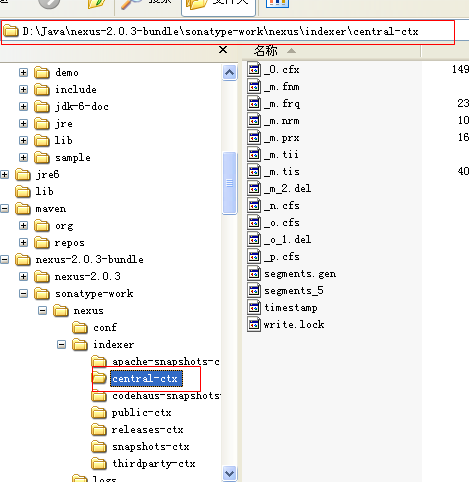
3、group仓库。组仓库用来方便开发人员进行设置的仓库

3、私有仓库的设置

1、更新私有仓库的索引

graphic

或者直接下载索引，将其拷贝到中央仓库的索引存储中



2、设置镜像（设置setting.xml）

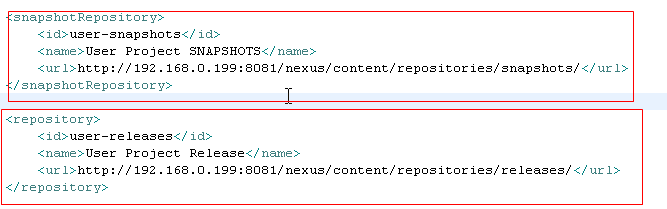


3、在settings.xml配置仓库，此时本机中的所有maven项目都会去私有仓库中查询依赖

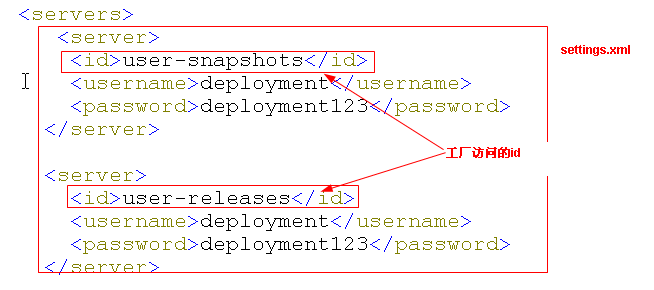


4、项目的发布

1、设定release工厂和snapshots工厂



2、设置访问的权限



5、创建项目工厂和设置权限

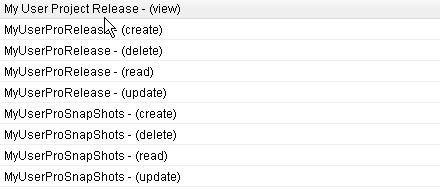
1、创建两个工厂：release和policy的

graphic

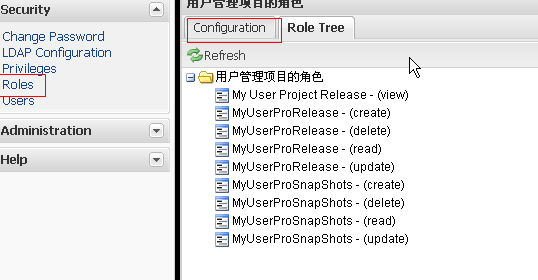
2、配置权限

3、

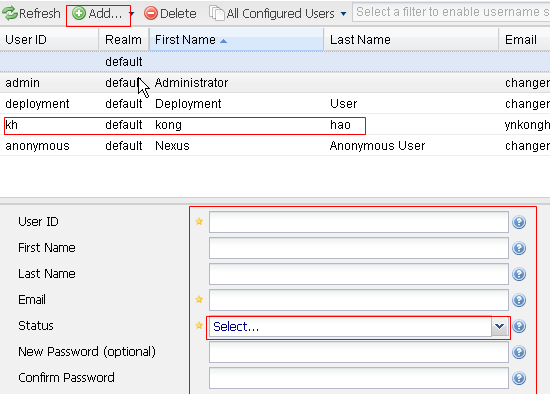




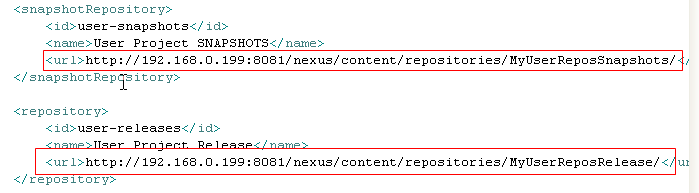
3、创建角色并且分配权限



4、创建用户



5、创建发布的工厂



**4、生命周期和插件**

生命周期

1、三套生命周期

1、clean

pre-clean 执行一些需要在clean之前完成的工作

clean 移除所有上一次构建生成的文件

post-clean 执行一些需要在clean之后立刻完成的工作

2、compile

validate

generate-sources

process-sources

generate-resources

process-resources 复制并处理资源文件，至目标目录，准备打包。

compile 编译项目的源代码。

process-classes

generate-test-sources

process-test-sources

generate-test-resources

process-test-resources 复制并处理资源文件，至目标测试目录。

test-compile 编译测试源代码。

process-test-classes

test 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。

prepare-package

package 接受编译好的代码，打包成可发布的格式，如 JAR 。

pre-integration-test

integration-test

post-integration-test

verify

install 将包安装至本地仓库，以让其它项目依赖。

deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

3、site

pre-site 执行一些需要在生成站点文档之前完成的工作

site 生成项目的站点文档

post-site 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备

site-deploy 将生成的站点文档部署到特定的服务器上

插件

插件是maven的核心，所有执行的操作都是基于插件来完成的

为了让一个插件中可以实现众多的类似功能，maven为插件设定了目标，一个插件中有可能有多个目标

其实生命周期中的重要的每个阶段都是由插件的一个具体目标来执行的

**5、测试和web的发布**

**6、测试**

**7、站点发布**

**8、插件开发**

maven

spring mvc ,hibernate

SOAP-->webservice

CXF

REST