## 一 版本管理

### 1.1何为版本管理

Maven能够很智能的处理各种特殊版本,解析各个模块最新的快照。使用该版本应该永远只能定位到唯一的构建。不通的版本能够促进团队开发与交流。对外发布时，显然是一个稳定的版本了。稳定版本发布完之后就进入到下一个阶段，又进入快照版本的开发了。

### 1.2 maven版本号定义约定

版本例子1.3.4-beta-2

1 表示该版本的第一个重大版本

3 表示基于第一个重大版本的第三个次要版本

4 表示该次要版本的第四个增量

beta-2 表示该增量的某一个里程碑

即 <主版本>.<次版本>.<增量版本>.<里程碑版本>

主版本：表示系统的重大架构变更 如struts1到struts2的变更

次版本: 表示较大范围功能增加与变化

增量版本: 表示一个重大bug问题的修复

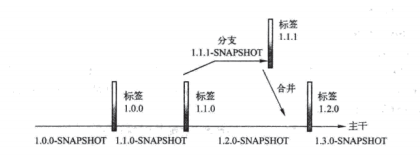
里程碑版本: 就是某一个非常不稳定的版本，需要很多的测试

### 1.3 maven主干，标签与分支

主干: 项目开发代码的主题,是从项目开始到现在都处于活动的状态。从这里可以获得到代码，以及几乎所有的变更历史。

标签:从主干的某个点分离出来的代码拷贝,通常在不影响主干的前提下，进行重大bug的修复，或者做一些实验性的开发。

分支:用来标识主干与分支的某个点的状态，代表某个代码的稳定状态。



如上图所示，展现了一个典型的项目版本管理的过程，展示了一个快照版本与稳定版本的切换，maven版本号约定的应用，以及版本控制系统主干，标签与分支的使用。

## 二 灵活构建

### 2.1常用内置属性

${baseDir} 根目录，pom.xml的文件目录

${version} 项目版本

### 2.2常用pom属性

${project.build.sourceDirectory} 项目主源码目录 src/main/java

${project.build.testSourceDirectory} 项目的测试源码目录/src/test/java

${project.build.resources} 项目主资源目录 src/main/resources

${project.build.testResources} 项目测试资源目录 src/main/testResources

${project.build.directory} 项目的构建输出目录 target/

${project.outputDirectory} 项目的主代码编译输出目录 target/classes/

${project.testOutputDirectory} 项目测试代码编译输出目录 /target/test/classes/

${project.groupId}项目groupId

${project.artifactId}项目artifactId

${project.version}项目version

${project.build.finalName}项目，项目打包输出文件的文件名默认

${project.artifactId}-${project.version}

### 2.3构建环境

#### 2.3.1 profile

jdbc.driverClassName=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://192.168.1.253:3306/ssh\_annotaction?useUnicode=true&characterEncoding=utf-8

jdbc.username=root

jdbc.password=123456

如上所示是数据库的链接配置，通常我们的做法是，要连接开发环境的时候，用开发环境的配置。要链接测试环境的时候用测试环境的配置。然后手动修改打包并发布。这是可行的，也是最常见的，然而这肯定不是最好的。手动意味着什么大家都知道。为了应对环境的变化，可以利用资源的过滤技术来处理。

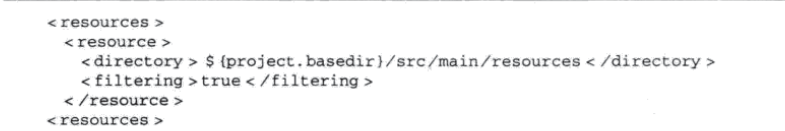
jdbc.driverClassName=${db.driver}

jdbc.url=${db.url}

jdbc.username=${db.username}

jdbc.password=${db.password}

通过对maven-resources-plugin进行如下配置





Mvn 的-P表示在命令行激活一个profile

<profiles>

<profile>

<!-- 开发环境 -->

<id>dev</id>

<properties>

<db.driver>1</db.driver>

<db.url>2</db.url>

<db.username>3</db.username>

<db.password>4</db.password>

</properties>

</profile>

<profile>

<!-- 测试环境 -->

<id>test</id>

<properties>

<db.driver>1</db.driver>

<db.url>2</db.url>

<db.username>3</db.username>

<db.password>4</db.password>

</properties>

</profile>

</profiles>

如上片段所示，是setting.xml文件中对profile的定义，只要通过-P,命令进行激活就可以进行灵活的构建了。

#### 2.3.2 profile的种类

Pom.xml中声明的profile只对当前项目有效。

Setting.xml中配置的profile对所有的maven项目都有效。

## 三maven插件

### 3.1 maven插件开发步奏

1 创建一个maven-plugin项目，插件本身也是一个maven项目。特殊的地方在于它的包必须是maven-plugin。可以通过eclipse快速创建一个代码生成器。

2 每个插件都必须包含一个或者多个目标,maven程之为Mojo(Maven old java Object)，编写插件的时候必须提供一个或者多个继承自AbstractMojo的类。

3 为目标提供配置点，大部分maven插件及其目标都是可配置的，因此在编写maven插件的时候注意提供可配置参数。

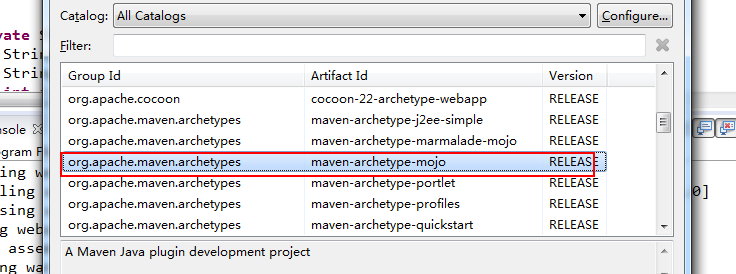
4 编写代码实现目标行为,根据实际的需要实现mojo

5 在代码中编写足够的错误日志，以便为用户提供足够的错误信息。

6 编写测试代码测试插件，然后再运行验证插件。

### 3.2开发maven插件

#### 3.2.1 创建插件项目



#### 3.2.2 查看插件项目pom.xml

<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/maven-v4\_0\_0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>cn.com.pconline</groupId>

<artifactId>maven-plugin</artifactId>

<packaging>maven-plugin</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>maven-plugin Maven Mojo</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

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

<artifactId>maven-plugin-api</artifactId>

<version>2.0</version>

</dependency>

<dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<!--This plugin's configuration is used to store Eclipse m2e settings only. It has no influence on the Maven build itself.-->

<plugin>

<groupId>org.eclipse.m2e</groupId>

<artifactId>lifecycle-mapping</artifactId>

<version>1.0.0</version>

<configuration>

<lifecycleMappingMetadata>

<pluginExecutions>

<pluginExecution>

<pluginExecutionFilter>

<groupId>

org.apache.maven.plugins

</groupId>

<artifactId>

maven-plugin-plugin

</artifactId>

<versionRange>

[3.2,)

</versionRange>

<goals>

<goal>descriptor</goal>

</goals>

</pluginExecutionFilter>

<action>

<ignore></ignore>

</action>

</pluginExecution>

</pluginExecutions>

</lifecycleMappingMetadata>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

注意这里引入maven-model的包,有些IDE是不会自动带入这个jar的，如果你需要${project.resources}这样的目录映入maven的资源文件路径的话,如果不需要的话就没必要引入。

<dependency>

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

<artifactId>maven-model</artifactId>

<version>3.2.2</version>

</dependency>

#### 3.2.3 编写maven插件

/\*\*

\* Goal which touches a timestamp file.

\* **@param** <Resource>

\*

\* **@goal** count

\*

\* **@phase** compile

\*/

**public** **class** CountLineMojo **extends** AbstractMojo

{

/\*\*

\* **@parameter**

\*/

**private** String[] includes;

/\*\*

\* **@parameter**

\*/

**private** String[] ratios;//**TODO** 定义为double[],从xml读取时提示java.lang.ClassCastException: [D cannot be cast to [Ljava.lang.Object;

**private** Map<String, Double> ratioMap = **new** HashMap<String, Double>();

**private** **long** realTotal;

**private** **long** fakeTotal;

**public** **void** execute() **throws** MojoExecutionException, MojoFailureException {

getLog().info("hello world plugin");

}

}

如上所示 注意点设置 @goal 继承AbstractMojo 其次是重写execute()方法，这三点是编写插件必须的，其他的字段自己可以看着加，需要什么字段自己加就好了。

#### 3.2.4 项目中引入插件

<build>

<plugins>

<plugin>

<groupId>cn.com.pconline</groupId>

<artifactId>maven-plugin</artifactId>

<version>0.0.1-SNAPSHOT</version>

<configuration>

<includes>

<includes>xxxxx</includes>

<includes>xxxxx</includes>

</includes>

</configuration>

<executions>

<execution>

<id>count line</id>

<phase>install</phase>

<goals>

<goal>

count

</goal>

</goals>

</execution>

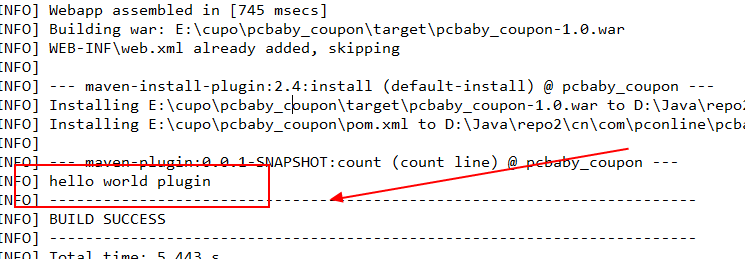
</executions>

</plugin>

</plugins>

</build>

如上代码片段展示了项目中如何引入插件。指定goal,执行阶段，执行参数。



如上是插件运行结果截图。

源代码参考 [git@github.com:wornxiao/framwork\_bymyself.git](mailto:git@github.com:wornxiao/framwork_bymyself.git) 已经上传到github

### 3.3 常用Mojo标注

@goal 插件目标名，识别并执行插件的插件标识。

@phase 插件生命周期阶段，就是指定插件执行时间。

@requiresProjefct true/false 指定这个插件是单独执行还是依赖项目执行，true标识依赖项目执行，false表示独立运行。

@aggregator 当Mojo在多模块项目上运行时，使用该标注，表示该目标只能在顶层模块上运行。

### 3.4 常用Mojo参数

#### 3.4.1 @paramter注解的方式

##### 3.4.1 Int

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** int phone;

<phone>xxxx</phone>

##### 3.4.2 Float

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** Float phone;

<phone>xxxx</phone>

##### 3.4.3 String

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** String phone;

<phone>xxxx</phone>

##### 3.4.**5** Date

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** String phone;

<phone>xxxx</phone>

##### 3.4.6 File

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** String phone;

<phone>c/fds/xxxx</phone>

##### 3.4.7 URL

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** String phone;

<phone>xxxx</phone>

##### 3.4.8 数组

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** String[] includes;

<includes>

<include>1</include>

<include>2</include>

</includes>

##### 3.4.**9** List

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** List<String> includes;

<includes>

<include>1</include>

<include>2</include>

</includes>

##### 3.4.**10** Map

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** Map<String> maps;

<maps>

<key1>1</key1>

<key2>2</key2>

</maps>

##### 3.4.11 **Properties**

/\*\*

\* **@parameter**

\*/

@Column(name = "phone")

**private** Map<String> maps;

<maps>

<property>

<name>1</name>

<value>2</value>

</property>

</maps>

#### 3.4.2 @readonly

只读的属相，用户不需要在项目中对它进行配置

#### 3.4.3 @require

必须的，也就是要求的，如果在项目中依赖了该插件，那么就必须配置这个参数，不然maven默认会报错。

### 3.5 Mojo 错误日志

getLog().info("TOTAL LINES:"+fakeTotal+ " ("+realTotal+")"); 可以通过getLog打日志的。