Лабораторная №7

Тема: «Использование плагинов в Maven»

Цель

Целью настоящей лабораторной работы является знакомство и освоение практических навыков использования плагинов в системе сборки проектов Maven.

Задание

В рамках лабораторной работы требуется дополнить проект maven из прошлой лабораторной работы 3-мя плагинами и продемонстрировать их работу. Список предлагаемых плагинов приведён в конце документа.

Содержание отчёта

Отчёт о выполнении работы должен включать в себя:

- титульный лист;
- вариант задания;
- краткое описание результатов

Теория

Использование плагина

В простейшем случае запустить плагин просто, например:

```
mvn org.apache.maven.plugins:maven-checkstyle-plugin:check
```

В данном примере вызывается плагин с

- groupId "org.apache.maven.plugins"
- artifactId "maven-checkstyle-plugin"
- последней версией
- целью (goal) "check"

Цель - это действие, которое плагин может выполнить. Целей может быть несколько.

плагины с groupId "org.apache.maven.plugins" можно запустить в более краткой форме:

```
mvn maven-checkstyle-plugin:check
```

или даже так:

```
mvn checkstyle:check
```

Объявление плагина в pom.xml

Объявление плагина похоже на объявление зависимости. Также, как и зависимости плагины идентифицируется с помощью GAV(groupId,artifactId,version). Например:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.6</version>
</plugin>
```

Объявление плагина в pom.xml позволяет зафиксировать версию плагина, задать ему необходимые параметры, привязать к фазам.

Привязка к фазам сборки проекта

После того как плагин объявлен, его можно настроить так, чтобы он автоматически запускался в нужный момент. Это делается с помощью привязки плагина к фазе сборки проекта:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.6</version>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

в данном примере плагин запустится в фазе проекта package

Настройки

Для работы большинства плагинов обычно требуются дополнительные настройки, которые специфичны для конкретного плагина. Настройки задаются в тэгах <configuration>. Например так настраивается tomcat - плагин:

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>tomcat-maven-plugin</artifactId>
    <version>1.1</version>
    <configuration>
        <fork>false</fork>
        <server>test-server</server>
        <url>http://test-server/manager</url>
    </configuration>
</plugin>
```

Содержимое в тэгах зависит от конкретного плагина и описывается в документации по плагину.

Примеры плагинов maven

maven-compiler-plugin

Компилятор - основной плагин который используется практически во всех проектах. Он доступен по умолчанию, но практически в каждом проекте его приходится переобъявлять т.к. настройки по умолчанию не очень подходящие.

Пример использования:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.0.2</version>
    <configuration>
        <source>1.6</source>
        <target>1.6</target>
        <encoding>UTF-8</encoding>
    </configuration>
</plugin>
```

В этом примере в кофигурации используется версия java 1.6 (source - версия языка на котором написана программа; target - версия java машины которая будет этот код запускать) и указано что кодировка исходного кода программы UTF-8. По умолчанию версии java - 1.3 а кодировка - та которая у операционной системы по умолчанию.

Вообще у плагина есть две цели compiler:compile и compiler:testCompile

- **compiler:compile** - компилирует основную ветку исходников и по умолчанию связана с фазой compile
- **compiler:testCompile** - компилирует тесты и по умолчанию связана с фазой test-compile.

Кроме приведёных настроек для компилятора можно задать следующие параметры:

- **verbose** true или false
- **fork** запусть компиляцию в отдельной jvm
- **executable** путь к javac
- **compilerVersion**
- **meminitial**
- **maxmem**
- **debug**
- **compilerArgument** задать аргуметы в одной коммандной строке-verbose -bootclasspath ${java.home}\lib\rt.jar
- **compilerArguments** задать аргуметы в коммандной строки пораздельно в тегах verbose, bootclasspath и др.
- **compilerId** позволяет задать язык программирования исходного кода, например csharp

maven-surefire-plugin

maven-surefire-plugin - плагин который запускает тесты и генерирует отчёты по результатам их выполнения. По умолчанию отчёты сохраняются в ${basedir}/target/surefire-reports и находятся в двух форматах - txt и xml. maven-surefire-plugin содержит единственную цель surefire:test тесты можно писать используя как JUnit так и TestNG.

По умолчанию запускаются все тесты с такими именами * "**/Test*.java" - включает все java файлы которые начинаются с "Test" и расположены в поддиректориях. * "**/*Test.java" - включает все java файлы которые заканчиваются на "Test" и расположены в поддиректориях. * "**/*TestCase.java" - включает все java файлы которые заканчиваются на "TestCase" и расположены в поддиректориях.

Чтобы вручную добавлять или удалять классы тестов можно посмотреть здесь http://maven.apache.org/plugins/maven-surefire-plugin/examples/inclusion-exclusion.html.

Запустить отдельный тест можно так: mvn -Dtest=TestCircle test имейте в виду что если вы хотите отладить тест в среде разработки то в конфигурации плагина нужно выставить

```
<forkMode>never</forkMode>
```

либо запускать тесты с remotedebug

```
mvn -Dmaven.surefire.debug test
```

Пропустить выполнение тестов можно

```
                    <configuration>
                      <skipTests>true</skipTests>
                    </configuration>
```

или

```
mvn install -DskipTests
```

чтобы пропустить ещё и компиляцию тестов вызовите maven так:

```
mvn install -Dmaven.test.skip=true
```

maven-source-plugin

Самый простой способ- создание архива - выполнить в командной строке :

```
mvn source:jar
```

Чтобы этот плагин автоматически запускался во время сборки плагина рекомендуется добавить его в pom.xml в раздел /project/build/plugins/ http://open.bekk.no/keeping-your-maven-build-fast/

```
 <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-source-plugin</artifactId>
    <version>2.1.2</version>
    <executions>
        <execution>
            <id>attach-sources</id>
            <phase>verify</phase>
            <goals>
                <goal>jar</goal>
            </goals>
```

```
        </execution>
    </executions>
</plugin>
```
В этом случае он jar с исходниками будет собираться на фазе verify, потом инсталлироваться на фазе install в локальный репозиторий и деплоится на фазе deploy в корпоративный репозиторий.

maven-javadoc-plugin

Плагин maven-javadoc-plugin предназначен для того чтобы генерировать докуметацию по исходному коду проекта стандартной утилитой javadoc.

maven-checkstyle-plugin

Это очень полезный плагин. Плагин проверяет стиль и качество исходного кода. Проверка качества кода особенно актуальна при разработке в команде из нескольких программистов. Автоматизация такой проверки - большая помощь в этой нудной и кропотливой работе.

Плагин основан на проекте http://checkstyle.sourceforge.net/. Из наиболее часто используемых и простых проверок:

- наличие комментариев
- размер класса не более N строк
- в конструкции в try-catch, блок catch не пустой.
- не используется System.out.println(.. вместо LOG.error(..

Подключить плагин довольно просто:

```
..
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>2.7</version>
        </plugin>
...........
```
после этого можно запустить проверку кода:

```
mvn checkstyle:check
```
Плагин проверит исходный код на наличие нарушений и сгенерирует файл checkstyle-result.xml. Чекстайл удобно использовать совместно с непрерывной интеграцией. Автоматическая проверка кода сильно экономит время. Самое важное - относиться к checkstyle ошибкам также как и ошибкам компиляции - при из возникновении сразу исправлять, т.к. когда ошибок накапливается сотни, их исправлять и тратить время хочется ещё меньше.. Если checkstyle ошибки исправляются как только они появятся- весь код будет чисто написан и комментирован, и можно быть больше уверенным в его качестве.

Т.к. почти каждые проекты пишутся немного по-разному, рекомедую создать свой набор правил. Полный набор правил описан тут: http://checkstyle.sourceforge.net/availablechecks.html и задать

его можно в специальном конфигурационном файле. (см. http://checkstyle.sourceforge.net/config.html) пример файла со сравнительно нестрогой проверкой приведён...

Внутри jar плагина есть примеры конфигурационных файлов:

- config/sun_checks.xml - от Sun Microsystems. Используются по умолчанию.
- config/maven_checks.xml - от Maven.
- config/turbine_checks.xml -от Turbine.
- config/avalon_checks.xml - от Avalon.

Если каком то месте кода появляется ошибка, но по объективным причинам код такой и должен быть, можно подавить вывод ошбки используя модуль SuppressionCommentFilter или SuppressionFilter

Пример объявления maven-checkstyle-plugin из реально работающего кода

```
 <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>2.6</version>
    <configuration>
        <configLocation>src/config/checkstyle.xml</configLocation>
            <consoleOutput>true</consoleOutput>
        </configuration>
        <executions>
            <execution>
                <phase>package</phase>
                <goals><goal>check</goal></goals>
            </execution>
        </executions>
</plugin>
```

maven-pmd-plugin

maven-pmd-plugin - плагин для автоматического анализа кода на предмет наличия "нехорошего кода". Также в этом плагине есть цель которая находит дубликаты кода Copy/Paste Detector (CPD). Основан на проекте a http://pmd.sourceforge.net/

Существует два режима работы плагина:

- генерирование отчётов PMD и CPD (цели pmd:pmd pmd:cpd) полезно для оценки качества существующих проектов которые раньше не использовали эти инструменты. Позволяет оценить масштабы "бедствия".
- проверяют проект на наличие нарушений. В случае, если находится нарушение, сборка ломается(цели pmd:check pmd:cpd-check). Удобно использовать в самом начале проекта совместно с непрерывной интеграцией. Гарантирует что код всегда "чистый". Экономит много времени ревизии кода (code review). Позволяет сделать разработку более масштабируемой - для большого проекта можно нанять больше программистов.

Пример декларирования плагина для второго случая:

```
 <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-pmd-plugin</artifactId>
    <version>2.5</version>
    <configuration>
        <targetJdk>1.6</targetJdk>
        <verbose>true</verbose>
    </configuration>
    <executions>
        <execution>
            <phase>package</phase>
            <goals><goal>check</goal></goals>
        </execution>
    </executions>
 </plugin>
```

findbugs-maven-plugin

findbugs-maven-plugin плагин для автоматического нахождения багов в проекте. Плагин основан на проекте http://findbugs.sourceforge.net/.

Принцип действия плагина основан на поиске шаблонов ошибок. Код проекта проверяются на часто встречаемые ошибки, неправильное использование API и конструкций языка.

Рекомендуется использовать в непрерывной интеграции совместно с maven-pmd-plugin и maven-checkstyle-plugin

maven-jar-plugin

Для сборки Jar-файла необходимо использовать соответствующий плагин.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <addMavenDescriptor>false</addMavenDescriptor>
            <compress>true</compress>
            <manifest>
                <addClasspath>true</addClasspath>
                <classpathPrefix>libs/</classpathPrefix>
                <mainClass>a1s.client.App</mainClass>
            </manifest>
        </archive>
    </configuration>
    <version>2.4</version>
</plugin>
```

В данном случае загружается плагин для сборки в jar-файл и производятся настройки:

- <addMavenDescriptor>false</addMavenDescriptor> - отключает копирование pom.xml и pom.properties в META-INF/maven
- <compress>true</compress> - использовать сжатие

- <addClasspath>true</addClasspath> - добавить зависимости в classpath манифеста
- <classpathPrefix>libs/</classpathPrefix> - путь к библиотекам зависимостей, которые добавляются в classpath
- <mainClass>a1s.client.App</mainClass> - полное имя класса, в котором используется метод main

maven-dependency-plugin

Копирование зависимостей

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
    <executions>
        <execution>
            <id>copy-dependencies</id>
            <phase>package</phase>
            <goals>
                <goal>copy-dependencies</goal>
            </goals>
            <configuration>
                    <outputDirectory>$
{project.build.directory}/libs</outputDirectory>
            </configuration>
        </execution>
    </executions>
    <version>2.5.1</version>
</plugin>
```

Данный плагин копирует файлы зависимостей в соответствующую директорию при сборке. Данный плагин реализует цель copy-dependencies, которая включается на этапе package. В качестве конфигурации передаётся директория для копирования файлов зависимостей ($ {project.build.directory}/libs), в нашем случае это будет target/libs.

jetty-maven-plugin

Embedded Jetty

```
<plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>8.1.8.v20121106</version>
    <configuration>
        <scanIntervalSeconds>10</scanIntervalSeconds>
        <stopKey>foo</stopKey>
        <stopPort>9999</stopPort>
        <connectors>
            <connector
implementation="org.eclipse.jetty.server.nio.SelectChannelConnector">
                <port>9090</port>
                <maxIdleTime>60000</maxIdleTime>
            </connector>
        </connectors>
    </configuration>
```

```
</plugin>
```

Рассмотрим настраиваемые свойства:

- scanIntervalSeconds - интервал в секундах через который Jetty проверяет директории на изменения для редеплоя приложения
- connectors - список подключений
- port - настраивает порт, на котором будет запущен jetty

После подключения плагина стало возможным проводить развёртывание проекта с использованием команды mvn jetty:run.

versions-maven-plugin

Проверка версий используемых зависимостей

Для maven есть плагин для проверки версий зависимостей. pom.xml

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>versions-maven-plugin</artifactId>
    <version>2.0</version>
</plugin>
```

```
$ mvn versions:display-dependency-updates
```

```
[INFO] The following dependencies in Dependencies are using the newest version:
[INFO]   javax.servlet:jstl ..................................... 1.2
[INFO]   joda-time:joda-time-jsptags ............................ 1.1.1
[INFO]   junit:junit ............................................ 4.11
[INFO]   org.apache.tiles:tiles-core ............................ 3.0.1
[INFO]   org.apache.tiles:tiles-jsp ............................. 3.0.1
[INFO]   org.eclipse.persistence:eclipselink .................... 2.3.2
[INFO]   org.eclipse.persistence:javax.persistence .............. 2.0.3
[INFO]   org.eclipse.persistence:org.eclipse.persistence.jpa.modelgen.processor ...
[INFO]                                                             2.3.2
[INFO]   org.hibernate:hibernate-annotations ................... 3.5.6-Final
[INFO]   org.springframework:spring-aop ..................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-beans ................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-context ................. 3.2.2.RELEASE
[INFO]   org.springframework:spring-core .................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-jdbc .................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-orm ..................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-test .................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-tx ...................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-web ..................... 3.2.2.RELEASE
[INFO]   org.springframework:spring-webmvc .................. 3.2.2.RELEASE
[INFO]   org.springframework.security:spring-security-config .... 3.1.3.RELEASE
[INFO]   org.springframework.security:spring-security-core ...... 3.1.3.RELEASE
[INFO]   org.springframework.security:spring-security-ldap ...... 3.1.3.RELEASE
[INFO]   org.springframework.security:spring-security-taglibs ... 3.1.3.RELEASE
[INFO]   org.springframework.security:spring-security-web ....... 3.1.3.RELEASE
[INFO]   taglibs:standard ....................................... 1.1.2
[INFO]
```

```
[INFO] The following dependencies in Dependencies have newer versions:
[INFO]   cglib:cglib ..................................... 2.2.2 -> 3.0
[INFO]   ch.qos.logback:logback-access ..................... 1.0.9 -> 1.0.11
[INFO]   ch.qos.logback:logback-classic .................... 1.0.9 -> 1.0.11
[INFO]   ch.qos.logback:logback-core ....................... 1.0.9 -> 1.0.11
[INFO]   com.google.guava:guava ........................... 14.0-rc3 -> 14.0.1
[INFO]   com.h2database:h2 ................................ 1.3.170 -> 1.3.171
[INFO]   javax.servlet:servlet-api ........................ 2.5 -> 3.0-alpha-1
[INFO]   javax.servlet.jsp:jsp-api ........................ 2.2 -> 2.2.1-b03
[INFO]   joda-time:joda-time ................................... 2.1 -> 2.2
[INFO]   mysql:mysql-connector-java ....................... 5.1.21 -> 5.1.24
[INFO]   net.sf.ehcache:ehcache ............................. 2.6.5 -> 2.7.0
[INFO]   org.apache.tomcat:tomcat-jdbc ..................... 7.0.37 -> 7.0.39
[INFO]   org.hibernate:hibernate-c3p0 .............. 4.2.0.Final -> 4.3.0.Beta1
[INFO]   org.hibernate:hibernate-core .............. 4.2.0.Final -> 4.3.0.Beta1
[INFO]   org.hibernate:hibernate-ehcache ........... 4.2.0.Final -> 4.3.0.Beta1
[INFO]   org.hibernate:hibernate-entitymanager ..... 4.2.0.Final -> 4.3.0.Beta1
[INFO]   org.hibernate:hibernate-validator ........... 4.3.1.Final -> 5.0.0.CR5
[INFO]   org.slf4j:jcl-over-slf4j ........................... 1.7.2 -> 1.7.5
[INFO]   org.slf4j:jul-to-slf4j ............................. 1.7.2 -> 1.7.5
[INFO]   org.slf4j:slf4j-api ................................ 1.7.2 -> 1.7.5
```

Из данного вывода можно увидеть какие версии и каких зависимостей поменялись.

joda-beans-maven-plugin

Генерация методов set, get, и пр. в моделях.

```
<build>
   <pluginManagement>
     <plugins>
       <plugin>
         <groupId>org.joda</groupId>
         <artifactId>joda-beans-maven-plugin</artifactId>
         <version>1.0</version>
       </plugin>
     </plugins>
   </pluginManagement>
 </build>
```