

Продвинутая сборка Maven-проекта

JSP & Servlets
2 уровень, 0 лекция

ОТКРЫТА

1.1 Список плагинов для сборки в Maven

Сборка в Maven может быть настроена очень гибко. Разработчики Maven специально создали десятки плагинов, используя которые можно очень гибко настраивать различные сборки. Самые популярные из них приведены в таблице ниже:

	Плагин	Описание
1	maven-compiler-plugin	Управляет Java-компиляцией
2	maven-resources-plugin	Управляет включением ресурсов в сборку
3	maven-source-plugin	Управляет включением исходного кода в сборку
4	maven-dependency-plugin	Управляет процессом копирования библиотек зависимостей
5	maven-jar-plugin	Плагин для создания итогового jar-файла
6	maven-war-plugin	Плагин для создания итогового war-файла
7	maven-surefire-plugin	Управляет запуском тестов
8	buildnumber-maven-plugin	Генерирует номер сборки

Каждый плагин по-своему интересен, но разобрать нам придется их все. Начнем мы с главного – плагина по управлению компиляцией.

1.2 Плагин компиляции maven-compiler-plugin

Самый популярный плагин, позволяющий управлять версией компилятора и используемый практически во всех проектах, – это компилятор `maven-compiler-plugin`. У него имеются настройки по умолчанию, однако практически в каждом проекте их нужно задать заново.

В самом простом варианте в плагине нужно задать версию исходного Java-кода и версию Java-машины, под которые осуществляется сборка:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.2</version>
  <configuration>
    <source>1.11</source>
    <target>1.13</target>
    <encoding>UTF-8</encoding>
  </configuration>
</plugin>
```

В примере выше мы задаем три параметра Java-компилятора: `source`, `target` и `encoding`.

Параметр `source` позволяет нам задать версию Java для наших исходников. Параметр `target` – версию Java-машины, под которую нужно скомпилировать классы. Если версия кода или Java-машины не задана, то по умолчанию используется параметр 1.3

Наконец параметр `encoding` позволяет указать кодировку Java-файлов. Мы указали `UTF-8`. Сейчас практически все исходники хранятся в кодировке `UTF-8`. Но если этот параметр не указан, то выберется текущая кодировка операционной системы. Для Windows – это кодировка `Windows-1251`.

Также бывают случаи, когда компьютер, на котором производится сборка имеет несколько установленных версий Java: для сборки разных модулей и/или разных проектов. В этом случае в переменной `JAVA_HOME` может быть указан только путь к одной из них.

Кроме того, бывают разные реализации Java-машины: OpenJDK, OracleJDK, Amazon JDK. И чем больше проект, тем сложнее его структура. Но вы можете явно задать плагину путь к компилятору `javac` с помощью тега `compilerPath`. Его добавили специально на этот случай.

Плагин `maven-compiler-plugin` имеет две цели (goals):

- `compiler:compile` – компиляция исходников, по умолчанию связана с фазой `compile`
- `compiler:testCompile` – компиляция тестов, по умолчанию связана с фазой `test-compile`.

Также можно указать список аргументов, которые будут переданы `javac`-компилятору в командной строке:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.2</version>
  <configuration>
    <compilerArgs>
      <arg>-verbose</arg>
      <arg>-Xlint:all, -options, -path<arg>
    </compilerArgs>
  </configuration>
</plugin>
```

1.3 Плагин создания jar-файла maven-jar-plugin

Если вы захотите собрать с помощью Maven свою собственную jar-библиотеку, то вам понадобится плагин `maven-jar-plugin`. Этот плагин умеет много полезных вещей.

Пример такого плагина:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <includes>
      <include>*/properties/*</include>
    </includes>
    <archive>
      <manifestFile>src/main/resources/META-INF/MANIFEST.MF</manifestFile>
    </archive>
  </configuration>
</plugin>
```

Во-первых, с его помощью можно указать, какие файлы попадут в библиотеку, а какие – нет. С помощью тегов `<include>` в секции `<includes>` можно задать **список директорий, чей контент нужно добавить в библиотеку**.

Во-вторых, каждая jar-библиотека должна иметь манифест (файл **MANIFEST.MF**). Плагин сам положит его в нужное место библиотеки, вам всего лишь нужно указать, по какому пути его взять. Для этого используется тег `<manifestFile>`.

И наконец, плагин может самостоятельно сгенерировать манифест. Для этого вместо тега `<manifestFile>` вам нужно добавить тег `<manifest>` и в нем указать данные для будущего манифеста. Пример:

```
<configuration>
  <archive>
    <manifest>
      <addClasspath>true</addClasspath>
      <classpathPrefix>lib/</classpathPrefix>
      <mainClass>ru.javarush.MainApplication</mainClass>
    </manifest>
  </archive>
</configuration>
```

Тег `<addClasspath>` определяет необходимость добавления в манифест `CLASSPATH`.

Тег `<classpathPrefix>` позволяет дописывать префикс (в примере lib) перед каждым ресурсом. Определение префикса в `<classpathPrefix>` позволяет размещать зависимости в отдельной папке.

Да, вы можете размещать библиотеки внутри другой библиотеки. И вас ждет много сюрпризов, когда вам нужно будет куда-то передать путь к properties-файлу, который находится в jar-библиотеке, которая находится в jar-библиотеке.

И наконец, тег `<mainClass>` указывает на главный исполняемый класс. “Что за **главный исполняемый класс?**”, – спросите вы. А все дело в том, что Java-машина может запустить программу, которая задана не только Java-классом, но и jar-файлом. И именно для такого случая нужен главный стартовый класс.

1.4 Плагин генерации номера сборки buildnumber-maven-plugin

Очень часто jar-библиотеки и war-файлы включают в себя информацию с названием проекта и его версией, а также версией сборки. Мало того, что это полезно для управления зависимостями, так еще и упрощает тестирование: понятно, в какой версии библиотеки ошибка исправлена, а в какой – добавлена.

Чаще всего эту задачу решают так – создают специальный файл `application.properties`, который содержит всю нужную информацию и включают его в сборку. Так же можно настроить сценарий сборки так, чтобы данные из этого файла перекочевывали в `MANIFEST.MF` и тому подобное.

Но что самое интересное, так это то, что у Maven есть специальный плагин, который может генерировать такой application.properties файл. Для этого нужно создать такой файл и заполнить его специальными шаблонами данных. Пример:

```
# application.properties
app.name=${pom.name}
app.version=${pom.version}
app.build=${buildNumber}
```

Значения всех трех параметров будут подставляться на этапе сборки.

Параметры `pom.name` и `pom.version` будут браться прямо из `pom.xml`. А для генерации уникального номера сборки в Maven есть специальный плагин – `buildnumber-maven-plugin`. Смотрите пример ниже:

```
<packaging>war</packaging>
<version>1.0</version>

<plugins>
  <plugin>
```

```
<groupId>org.codehaus.mojo</groupId>
<artifactId>buildnumber-maven-plugin</artifactId>
<version>1.2</version>
<executions>
  <execution>
    <phase>validate</phase>
    <goals>
      <goal>create</goal>
    </goals>
  </execution>
</executions>
<configuration>
  <revisionOnScmFailure>true</revisionOnScmFailure>
  <format>{0}-{1,date,yyyyMMdd}</format>
  <items>
    <item>${project.version}</item>
    <item>timestamp</item>
  </items>
</configuration>
</plugin>
</plugins>
```

В примере выше происходят три важные вещи. Во-первых, указан сам плагин для задания версии сборки. Во-вторых, указано, что он будет запускаться во время фазы `validate` (самая первая фаза) и генерировать номер сборки – `${buildNumber}`.

И в-третьих, **указан формат этого номера сборки, который склеивается из нескольких частей**. Это версия проекта `project.version` и текущее время заданное шаблоном. Формат шаблона задается Java-классом `MessageFormat`.

+21

Комментарии (8)

популярные

новые

старые

JavaCoder

Введите текст комментария

Ильгиз

Уровень 41, Уфа, Россия

21 октября, 09:47

как потом использовать сгенерированный плагином buildnumber-maven-plugin номер сборки? куда он записывается и как его отображать к примеру в приложении?

Ответить

0

Denys D.

Уровень 79, Киев, Украина

14 июля, 09:00

какие файлы попадут с библиотеку, а какие – нет.

Опечатка

Ответить

+3

Mentor-02

Backend Developer в JavaRush

MENTOR

31 октября, 18:33

Спасибо, поправили

Ответить

0

Андрей

Уровень 68, Винница, Украина

13 июня, 14:28

Нашел две орфографические ошибки (буква і не там):

Для этого используется тег <mainfestFile>.

И наконец, плагин может самостоятельно сгенерировать манифест. Для этого вместо тега <manifestFile> вам нужно добавить тег <mainfest> и в нем указать данные для будущего манифеста.

Ответить

+2

Руслан

Уровень 43

15 июня, 15:33

Как получить 60-й уровень?)

Ответить

+1

Александр Огарков

Уровень 71, Мытищи, Россия

10 июля, 00:56

JRU

Ответить

0

Андрей Пазюк

Уровень 69, Винница, Украина

10 июля, 20:51

Руслан, вот так.

Ответить

+5

Mentor-02

Backend Developer в JavaRush

MENTOR

31 октября, 18:33

Спасибо, поправили

Ответить

0

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ



