Управление файлами во время сборки Maven-проекта

JSP & Servlets
2 уровень, 1 лекция

ОТКРЫТА

2.1 Плагин копирования ресурсов maven-resources-plugin

Если ты собираешь web-приложение, то у тебя будет просто куча различных ресурсов в нем. Это jar-библиотеки, jsp-сервлеты, файлы настроек. Ну и конечно же это куча статических файлов типа `html`, `css`, `js`, а также различных картинок.

По умолчанию Maven при сборке проекта просто скопирует все ваши файлы из папки `src/main/resources` в директорию target. Если же ты хочешь внести изменения в это поведение, то вам поможет плагин `maven-resources-plugin`.

Пример кода такого плагина:

```
<plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.6</version>
    <executions>
        <execution>
            <id>copy-resources</id>
            <phase>validate</phase>
            <goals>
                <goal>copy-resources</goal>
            </goals>
            <configuration>
                <outputDirectory>
                   ${basedir}/target/resources
                </outputDirectory>
                <resources>
                    <resource>  инструкции по копированию ресурса 1 </resource>
                    <resource>  инструкции по копированию ресурса 2 </resource>
                    <resource>  инструкции по копированию ресурса N </resource>
                </resources>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Данный плагин вызовется во время фазы validate. С помощью тега `<outputDirectory>` можно задать директорию, в которую плагин должен будет скопировать ресурсы, заданные в секции `<resources>`. И вот тут-то плагин может развернуться во всю свою мощь.

2.2 Фильтрация ресурсов с помощью maven-resources-plugin

Ресурсы плагина можно задавать не только в виде файлов, а сразу в виде директорий. Более того, к директории можно добавить маску, которая задает какие именно файлы из нее будет включены в данный ресурс.

Пример:

```
            <resource>
```

```
            <directory>src/main/resources/images</directory>
            <includes>
                 <include>**/*.png</include>
            </includes>
        </resource>
```

Две звездочки в качестве маски обозначают **любое количество директорий**. В примере выше в качестве данных ресурса будут взяты все png-файлы, которые содержаться в директории `src/main/resources/images` (и ее поддиректориях).

Если ты хочешь исключить какие-нибудь файлы, можешь воспользоваться тегом `exclude`. Пример:

```
<resource>
    <directory>src/main/resources/images</directory>
    <includes>
        <include>**/*.png</include>
    </includes>
    <excludes>
         <exclude>old/*.png</exclude>
    </excludes>
</resource>
```

Теги применяются последовательно: сначала к ресурсу будут добавлены указанные в include-файлы, а затем из этого списка исключат exclude-файлы.

Но и это еще не все. Плагин умеет заглядывать внутрь файлов (если они текстовые, конечно). И, например, добавить в файл `application.properties` нужную версию сборки. Чтобы плагин обрабатывал содержимое файла, нужно указать ему параметр `<filtering>true</filtering>`.

Пример:

```
<resource>
    <directory>src/main/resources/properties</directory>
    <filtering>true</filtering>
    <includes>
        <include>**/*. properties </include>
    </includes>
</resource>
```

Более подробно с данным плагином можно ознакомиться по ссылке: https://maven.apache.org/plugins/maven-resources-plugin/examples/filter.html

2.3 Плагин включения исходных кодов maven-source-plugin

Еще один полезный плагин – `maven-source-plugin` позволяет включать в сборку исходный код ваших java-файлов. Зачем?

Все дело в том, что кроме web-приложений, с помощью Maven собирается очень большое количество библиотек. Очень много Java-проектов следуют концепции open-source и распространяются среди Java-сообщества со своими исходниками.

Зачем нужен отдельный плагин? Почему нельзя просто скопировать исходники и все?

Во-первых, в любом сложном проекте исходники могут храниться в нескольких местах.

Во-вторых, часто используется генерация исходников на основе xml-спецификаций, такие исходники тоже нужно включать в сборку.

Ну и в-третьих, ты можешь решить не включать какие-нибудь особо секретные файлы в вашу сборку.

Пример использования плагина maven-source-plugin:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-source-plugin</artifactId>
    <version>2.2.1</version>
    <executions>
        <execution>
            <id>attach-sources</id>
            <phase>verify</phase>
            <goals>
                <goal>jar</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

2.4 Плагин копирования зависимостей maven-dependency-plugin

Также тебе может понадобиться умное копирование зависимостей (библиотек) при сборке проекта. Для этого используется плагин `maven-dependency-plugin`.

Пример:

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

В этом примере прописано дефолтное поведение плагина – копирование библиотек в директорию `${project.build.directory}/lib`.

В секции execution прописано, что плагин будет вызван во время фазы сборки – package, goal – copy-dependences.

В целом, у этого плагина довольно большой набор целей, вот самые популярные из них:

1	dependency:analyze	анализ зависимостей (используемые, неиспользуемые, указанные, неуказанные)
2	dependency:analyze-duplicate	определение дублирующихся зависимостей
3	dependency:resolve	разрешение (определение) всех зависимостей

4	dependency:resolve-plugin	разрешение (определение) всех плагинов
5	dependency:tree	вывод на экран дерева зависимостей

Также в разделе configuration можно задать дополнительные параметры:

1	outputDirectory	Директория, в которую будут копироваться зависимости
2	overWriteReleases	Флаг необходимости перезаписывания зависимостей при создании релиза
3	overWriteSnapshots	Флаг необходимости перезаписывания неокончательных зависимостей, в которых присутствует SNAPSHOT
4	overWriteIfNewer	Флаг необходимости перезаписывания библиотек с наличием более новых версий

Пример:

```
<configuration>
    <outputDirectory>
         ${project.build.directory}/lib/
    </outputDirectory>
    <overWriteReleases>false</overWriteReleases>
    <overWriteSnapshots>false</overWriteSnapshots>
    <overWriteIfNewer>true</overWriteIfNewer>
 </configuration>
```

По умолчанию `<overWriteReleases>` и `<overWriteSnapshots>` – false, для `<overWriteIfNewer>` – true.

‹ Предыдущая лекция

Следующая лекция ›

Рогов Игорь Уровень 13, Самара, Russian Federation 6 июня, 12:22

<filtering>true<filtering> / не хватает. чуть выще вместо файлов написано фейлов

Ответить +2

Mentor-02 Backend Developer в **JavaRush** MENTOR 13 июня, 13:37

Спасибо, исправлено

Ответить 0