Maven Dependency Plugin

Maven Dependency Plugin は、Java プロジェクト用の広く使用されるビルド自動化ツールである Maven の強力なツールで、プロジェクトの依存関係を管理および操作するのを助けます。Maven プロジェクトの依存関係は、`pom.xml` ファイルに宣言された外部ライブラリまたはモジュールで、Maven はこれらをリポジトリからダウンロードし、コンパイル、実行、またはテスト中に利用可能にします。Dependency Plugin は、これらの依存関係を分析、コピー、リスト表示、またはその他の方法で処理するためのさまざまなゴール（タスク）を提供することで、この機能を拡張します。以下に、効果的に使用する方法を示します。

---

1. 使用法の概要　Maven Dependency Plugin は、2 つの主要な方法で使用できます：- **`pom.xml` ファイルに設定する**: これにより、特定のプラグインのゴールを Maven ビルドライフサイクルのフェーズ（例：`package`、`install`）にバインドし、ビルドプロセス中に自動的に実行できます。- **コマンドラインから直接ゴールを実行する**: これは、一時的なタスクや `pom.xml` を変更したくない場合に適しています。

プラグインは、以下の座標で識別されます：`groupId: org.apache.maven.plugins`、`artifactId: maven-dependency-plugin`。設定する際にはバージョン（例：`3.2.0`）を指定する必要がありますが、コマンドラインの使用では Maven が最新バージョンを解決することが多いです。

---

2. `pom.xml` にプラグインを追加する　ビルドプロセスの一部としてプラグインを使用するには、`pom.xml` の `<build><plugins>` セクションに追加します。以下は基本的な例です：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>3.2.0</version>
        </plugin>
    </plugins>
</build>
```

この設定で、ビルドライフサイクル中に特定のゴールを実行するように設定するには、`<executions>` ブロックを追加します。

---

3. 一般的なゴールとその使用方法　プラグインは、依存関係を管理するためのいくつかのゴールを提供します。以下に、最も一般的に使用されるものとその使用例を示します。

a. `copy-dependencies`

- **目的**: プロジェクトの依存関係を指定されたディレクトリにコピーします（例：`lib` フォルダにパッケージ化）。
- `pom.xml` **に設定する**: このゴールを `package` フェーズにバインドして、`mvn package` 中に依存関係をコピーします：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>3.2.0</version>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>${project.build.directory}/lib</outputDirectory>
                        <includeScope>runtime</includeScope>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

  - `${project.build.directory}/lib` はプロジェクトの `target/lib` に解決されます。
  - `<includeScope>runtime</includeScope>` は、`compile` と `runtime` スコープの依存関係のみをコピーし、`test` と `provided` を除外します。

- **コマンドライン**: `pom.xml` を変更せずに直接実行します：

```
mvn dependency:copy-dependencies -DoutputDirectory=lib
```

b. `tree`

- **目的**: 依存関係ツリーを表示し、すべての直接的および間接的な依存関係とそのバージョンを示します。これはバージョンの競合を特定するのに役立ちます。

- **コマンドライン**: 単に実行します：

  ```
  mvn dependency:tree
  ```

  これは、依存関係の階層的なビューをコンソールに出力します。

- `pom.xml` **に設定する**（任意）：このゴールをビルドフェーズ（例：`verify`）中に実行する場合は、`copy-dependencies` と同様に設定します。

c. `analyze`

- **目的**: 依存関係を分析して、以下のような問題を特定します：
  - 使用されているが宣言されていない依存関係。
  - 宣言されているが使用されていない依存関係。
- **コマンドライン**: 実行します：

  ```
  mvn dependency:analyze
  ```

  これは、コンソールにレポートを生成します。

- **注意**: このゴールは、複雑なプロジェクトの場合には追加の設定が必要になることがあります。

d. `list`

- **目的**: プロジェクトのすべての解決された依存関係をリスト表示します。
- **コマンドライン**: 実行します：

  ```
  mvn dependency:list
  ```

  これは、依存関係のフラットなリストを提供し、迅速な参照に役立ちます。

e. `unpack`

- **目的**: 特定の依存関係（例：JAR ファイル）の内容をディレクトリに展開します。
- **コマンドライン**: 特定のアーティファクトを展開する例：

  ```
  mvn dependency:unpack -Dartifact=groupId:artifactId:version -DoutputDirectory=target/unpacked
  ```

  `groupId:artifactId:version` を依存関係の座標（例：`org.apache.commons:commons-lang3:3.12.0`）に置き換えます。

f. `purge-local-repository`

- **目的**: 指定された依存関係をローカル Maven リポジトリ（`~/.m2/repository`）から削除し、リモートリポジトリから新しいダウンロードを強制します。
- **コマンドライン**: 実行します：

  ```
  mvn dependency:purge-local-repository
  ```

  これは、破損した依存関係ファイルのトラブルシューティングに役立ちます。

---

4. カスタマイズオプション 多くのゴールは、その動作を調整するための設定パラメータをサポートしています：- `outputDirectory`: コピーまたは展開するファイルの場所を指定します（例：`target/lib`）。- **`includeScope` または** `excludeScope`: スコープ（例：`runtime`、`test`）で依存関係をフィルタリングします。- **`artifact`**: `unpack` などのゴールの対象となる特定の依存関係を指定します。

これらを `<configuration>` セクションの `pom.xml` またはコマンドライン引数として `-D`（例：`-DincludeScope=runtime`）で渡すことができます。

---

5. 実践的な例

- **シナリオ 1: 独立したアプリケーションのパッケージ化** `copy-dependencies` を使用して、すべてのランタイム依存関係を `lib` フォルダにまとめます：

  ```
  mvn dependency:copy-dependencies -DoutputDirectory=lib -DincludeScope=runtime
  ```

- **シナリオ 2: 依存関係の競合のデバッグ** `dependency:tree` を実行して、間接的な依存関係を検査します：

  ```
  mvn dependency:tree
  ```

  例えば、`commons-logging` の複数のバージョンを見つけた場合、`pom.xml` の `<dependencyManagement>` セクションに追加して競合を解決します。

- **シナリオ 3: リソースの展開** 依存関係を展開してその内容にアクセスします：

  ```
  mvn dependency:unpack -Dartifact=org.example:my-lib:1.0.0 -DoutputDirectory=target/resources
  ```

---

6. 要約　Maven Dependency Plugin を使用するには：- ビルドプロセスに組み込みたい場合は、`pom.xml` に特定のゴールと設定を追加します。- `copy-dependencies`、`tree`、`analyze`、`list`、`unpack` などのゴールをコマンドラインから直接実行して、アドホックタスクを行います。- 設定オプションを使用して、プロジェクトのニーズに合わせて動作をカスタマイズします。

このプラグインは、単なる宣言を超えた依存関係の管理に役立ち、ビルド、デバッグ、配布する Maven プロジェクトにとって非常に価値があります。