Maven 自动化构建 Maven 自动化部署

Maven 依赖管理

Maven 一个核心的特性就是依赖管理。当我们处理多模块的项目（包含成百上千个模块或者子项目），模块间的依赖关系就变得非常复杂，管理也变得很困难。针对此种情形，Maven 提供了一种高度控制的方法。

可传递性依赖发现

一种相当常见的情况，比如说 A 依赖于其他库 B。如果，另外一个项目 C 想要使用 A ，那么 C 项目也需要使用库 B。

Maven 可以避免去搜索所有所需库的需求。Maven 通过读取项目文件（pom.xml），找出它们项目之间的依赖关系。

我们需要做的只是在每个项目的 pom 中定义好直接的依赖关系。其他的事情 Maven 会帮我们搞定。

通过可传递性的依赖，所有被包含的库的图形会快速的增长。当有重复库时，可能出现的情形将会持续上升。Maven 提供一些功能来控制可传递的依赖的程度。

功能	功能描述
依赖调节	决定当多个手动创建的版本同时出现时，哪个依赖版本将会被使用。 如果两个依赖版本在依赖树里的深度是一样的时候，第一个被声明的依赖将会被使用。
依赖管理	直接的指定手动创建的某个版本被使用。例如当一个工程 C 在自己的依赖管理模块包含工程 B，即 B 依赖于 A， 那么 A 即可指定在 B 被引用时所使用的版本。
依赖范围	包含在构建过程每个阶段的依赖。
依赖排除	任何可传递的依赖都可以通过 "exclusion" 元素被排除在外。举例说明，A 依赖 B， B 依赖 C，因此 A 可以标记 C 为 "被排除的"。
依赖可选	任何可传递的依赖可以被标记为可选的，通过使用 "optional" 元素。例如：A 依赖 B， B 依赖 C。因此，B 可以标记 C 为可选的， 这样 A 就可以不再使用 C。

依赖范围

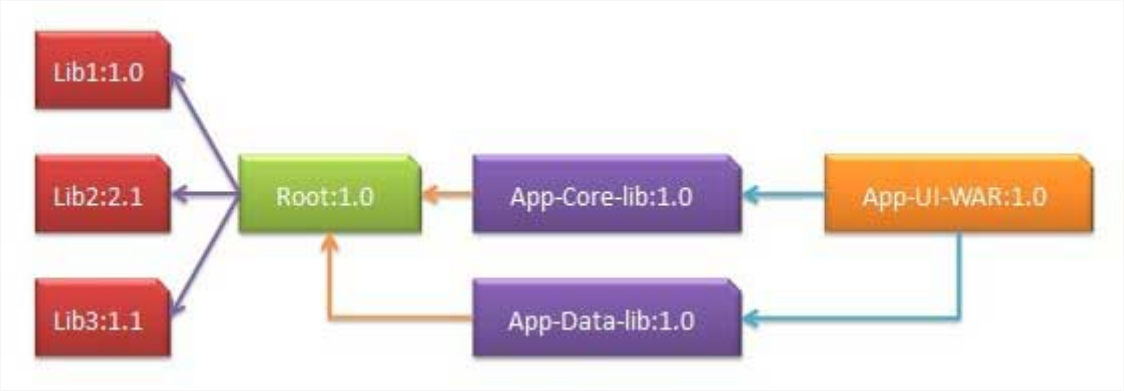
传递依赖发现可以通过使用如下的依赖范围来得到限制：

范围	描述
编译阶段	该范围表明相关依赖是只在项目的类路径下有效。默认取值。
供应阶段	该范围表明相关依赖是由运行时的 JDK 或者 网络服务器提供的。

范围	描述
运行阶段	该范围表明相关依赖在编译阶段不是必须的，但是在执行阶段是必须的。
测试阶段	该范围表明相关依赖只在测试编译阶段和执行阶段。
系统阶段	该范围表明你需要提供一个系统路径。
导入阶段	该范围只在依赖是一个 pom 里定义的依赖时使用。同时，当前项目的POM 文件的 部分定义的依赖关系可以取代某特定的 POM。

依赖管理

通常情况下，在一个共通的项目下，有一系列的项目。在这种情况下，我们可以创建一个公共依赖的 pom 文件，该 pom 包含所有的公共的依赖关系，我们称其为其他子项目 pom 的 pom 父。 接下来的一个例子可以帮助你更好的理解这个概念。



接下来是上面依赖图的详情说明：

- App-UI-WAR 依赖于 App-Core-lib 和 App-Data-lib。
- Root 是 App-Core-lib 和 App-Data-lib 的父项目。
- Root 在它的依赖部分定义了 Lib1、lib2 和 Lib3 作为依赖。

App-UI-WAR 的 pom.xml 文件代码如下：

```
<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/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.companyname.groupname</groupId>
<artifactId>App-UI-WAR</artifactId>
<version>1.0</version>
<packaging>war</packaging>
<dependencies>
```

```
<dependency>
<groupId>com.companyname.groupname</groupId>
<artifactId>App-Core-lib</artifactId>
<version>1.0</version>
</dependency>
</dependencies>
<dependencies>
<dependency>
<groupId>com.companyname.groupname</groupId>
<artifactId>App-Data-lib</artifactId>
<version>1.0</version>
</dependency>
</dependencies>
</project>
```

App-Core-lib 的 pom.xml 文件代码如下：

```
<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/xsd/maven-4.0.0.xsd">
<parent>
<artifactId>Root</artifactId>
<groupId>com.companyname.groupname</groupId>
<version>1.0</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<groupId>com.companyname.groupname</groupId>
<artifactId>App-Core-lib</artifactId>
<version>1.0</version>
<packaging>jar</packaging>
</project>
```

App-Data-lib 的 pom.xml 文件代码如下：

```
<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/xsd/maven-4.0.0.xsd">
<parent>
<artifactId>Root</artifactId>
<groupId>com.companyname.groupname</groupId>
<version>1.0</version>
</parent>
<modelVersion>4.0.0</modelVersion>
<groupId>com.companyname.groupname</groupId>
<artifactId>App-Data-lib</artifactId>
<version>1.0</version>
<packaging>jar</packaging>
</project>
```

Root 的 pom.xml 文件代码如下：

```
<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/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.companyname.groupname</groupId>
<artifactId>Root</artifactId>
<version>1.0</version>
<packaging>pom</packaging>
<dependencies>
<dependency>
<groupId>com.companyname.groupname1</groupId>
<artifactId>Lib1</artifactId>
<version>1.0</version>
</dependency>
</dependencies>
<dependencies>
<dependency>
<groupId>com.companyname.groupname2</groupId>
<artifactId>Lib2</artifactId>
<version>2.1</version>
</dependency>
</dependencies>
<dependencies>
<dependency>
<groupId>com.companyname.groupname3</groupId>
<artifactId>Lib3</artifactId>
<version>1.1</version>
</dependency>
</dependencies>
</project>
```

现在当我们构建 App-UI-WAR 项目时， Maven 将通过遍历依赖关系图找到所有的依赖关系，并且构建该应用程序。

通过上面的例子，我们可以学习到以下关键概念：

- 公共的依赖可以使用 pom 父的概念被统一放在一起。App-Data-lib 和 App-Core-lib 项目的依赖在 Root 项目里列举了出来（参考 Root 的包类型，它是一个 POM）.
- 没有必要在 App-UI-W 里声明 Lib1, lib2, Lib3 是它的依赖。 Maven 通过使用可传递的依赖机制来实现该细节。

← Maven 自动化构建 Maven 自动化部署 →