01 坐标和依赖

C-2 创建: 张林伟, 最后修改: 张林伟 2018-07-30 19:35

Maven坐标

Maven坐标的元素包括groupId、artifactId、version、packaging、classifier。

- groupId：定义当前maven项目隶属的实际项目。一个实际项目往往会被划分为多个模块。因为一个组织下会有很多实际项目，所以groupId不应该对应项目隶属的组织或公司。
- artifactId：定义实际项目中的一个maven项目（模块）。
- version：版本。
- packaging：打包方式。默认jar。
- classifier：用来帮助定义构建输出的一些附属构件。不能直接定义，需要附加的插件帮助生成。

依赖配置

代码块 XML

```xml
<project>
  ...
  <dependencies>
    ...
    <dependency>
      <groupId>...</groupId>
      <artifactId>...</artifactId>
      <version>...</version>
      <type>...</type>
      <scope>...</scope>
      <optional>...</optional>
      <exclusions>
        ...
        <exclusion>...</exclusion>
        ...
      </exclusions>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

- groupId、artifactId和version：依赖的坐标。
- type：依赖的类型，默认jar。
- scope：依赖的范围。
- optional：标记依赖是否可选。
- exclusions：用来排除传递性依赖。

依赖范围

实际上，编译项目主代码、编译和执行测试、实际运行maven项目会使用不用的classpath。依赖范围就是用来控制与这三种classpath（编译classpath、测试classpath、运行classpath）的关系。

- compile：编译依赖范围。默认值。使用此依赖范围的maven依赖，对于编译、测试、运行三种classpath都有效。
- test：测试依赖范围。只对测试classpath有效。
- provided：已提供依赖范围。对于编译和测试classpath有效，对运行classpath无效。
- runtime：运行时依赖范围。对于测试和运行classpath有效，对编译classpath无效。
- system：系统依赖范围。和provided依赖范围一致。但是system范围的依赖必须通过systemPath元素显示指导依赖文件的路径。这类依赖不通过maven仓库解析。
- import：导入依赖范围。不会对三种classpath产生影响。

传递性依赖

- 第二直接依赖范围是compile，传递性依赖范围与第一直接依赖范围一致。

- 第二直接依赖范围是test，依赖不会得以传递。
- 第二直接依赖范围是provided，只传递第一直接依赖范围也为provided的依赖，且传递性依赖范围同样为provided。
- 第二直接依赖范围是runtime，传递性依赖范围与第一直接依赖范围一致，但compile例外，此时传递性依赖范围为runtime。

依赖调解

A -> B -> C -> X (1.0)

A -> D -> X (2.0)

- 第一原则：路径最近者优先
- 第二原则：第一声明者优先

最佳实践

1.排除依赖：使用exclusions元素。

2.归类依赖：如spring家族

代码块 XML

```xml
<properties>
  <springframework.version>4.2.5.RELEASE</springframework.version>
</properties>
```

只需修改一处就可以全部升级

3.优化依赖

- mvn dependency:list 查看当前项目的已解析依赖（resolved dependency）
- mvn dependency:tree 查看当前项目的依赖树
- mvn dependency:analyze 分析当前项目的依赖，如哪些依赖声明但未使用，哪些使用但未声明