一： snapshot 版本滥用

- Snapshot 版本和 release 版本区别

如果是快照版本，那么在 mvn deploy 时会自动发布到快照版本库中，会覆盖老的快照版本，而在使用快照版本的模块，在不更改版本号的情况下，直接编译打包时，maven 会自动从镜像服务器上下载最新的快照版本。如果是正式发布版本，那么在 mvn deploy 时会自动发布到正式版本库中，而使用正式版本的模块，在不更改版本号的情况下，编译打包时如果本地已经存在该版本的模块则不会主动去镜像服务器上下载。

最佳实践：
开发环境用 snapshot 版本，线上版本用 release 版本。
线上环境禁用 snapshot 版本，snapshot 版本不稳定，传递依赖等导致系统 bug。

二：排除传递依赖

```
<dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
</dependency>
```

- 版本选择“最近路径优先原则”

项目 A

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

A->D->Z(2.0)

Z(1.0)的路径长度为 3

Z(2.0)的路径长度为 2

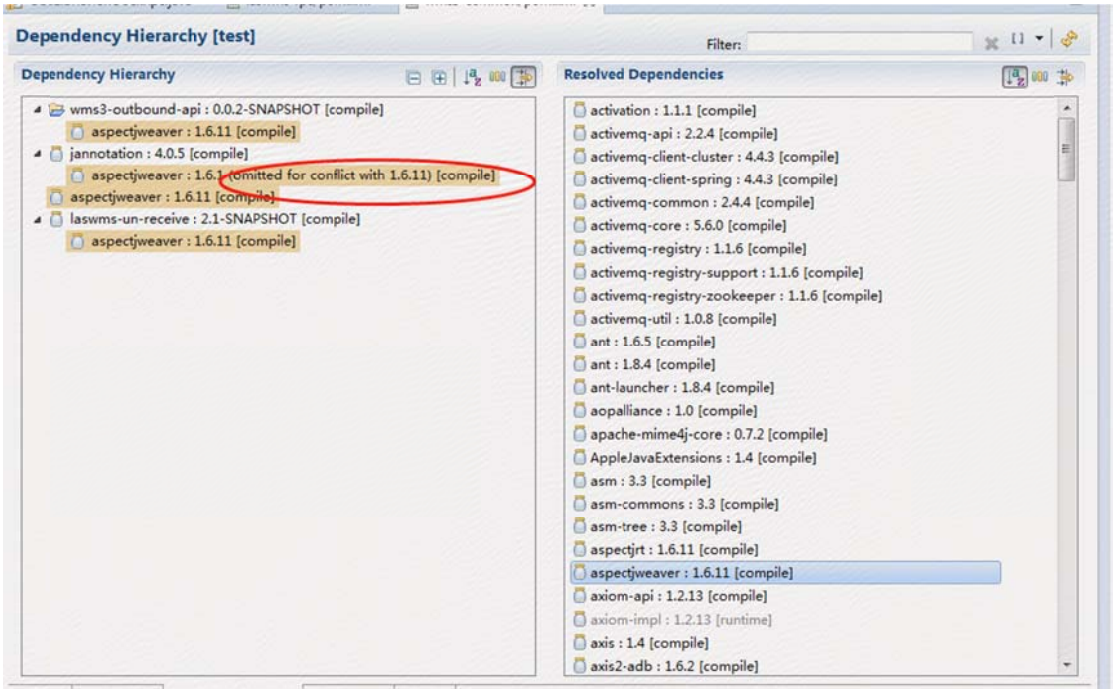
因此 Z(2.0)会被解析使用

如果路径相同，依据“最先引用，最先选择”原则。

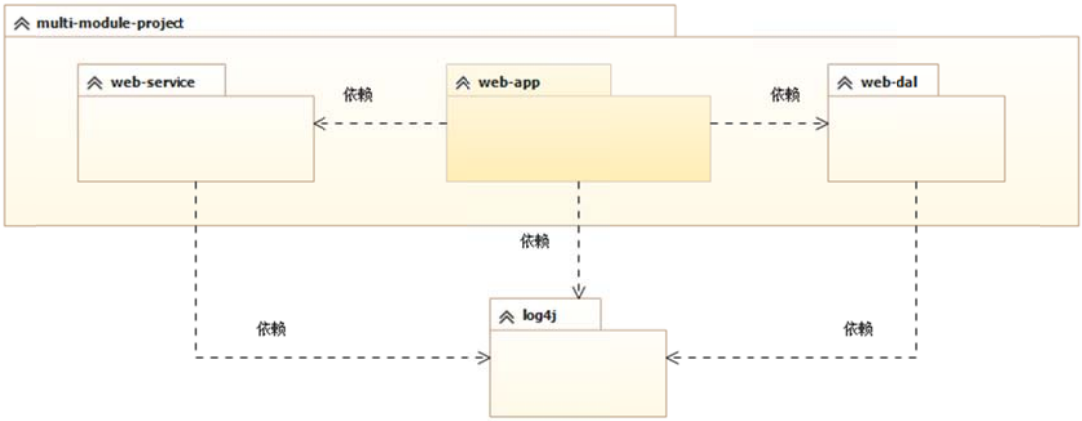
最佳实践：

如果版本引用不对，通过 mvn dependency:tree，使用该命令可以看到 maven 的依赖树

也可以利用 IDE 工具等查看版本依赖情况，然后进行排除操作



三．优化模块重复依赖



上图 log4j 的依赖关系比较复杂，存在多个模块共同引用的情况，不利于改动和扩展。

最佳实践：如果有两个以上的子模块共同依赖一个组件，那么对于这个组建的依赖就应该放在父模块中定义 dependencies。

四.利用 dependencyManagement 固化依赖配置

```
<dependencyManagement>
 <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactid>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactid>log4j</artifactId>
      <version>1.2.16</version>
    </dependency>
 </dependencies>
 </dependencyManagement>
```

父模块中 dependencyManagement 下的 junit 依赖不会对子模块产生任何影响。

如果某个子模块需要使用 JUnit 和 Log4j 的时候，我们就可以简化依赖配置成这样：

```
    <dependency>
        <groupId>junit</groupId>
        <artifactid>junit</artifactId>
      </dependency>
      <dependency>
        <groupId>log4j</groupId>
        <artifactid>log4j</artifactId>
    </dependency>
```

五．使用 import 单独出公共依赖

为了应对父模块中引入了大量的 jar 包依赖造成父模块臃肿，我们需要一种可以把 dependencyManagement 放到外面去分开管理，这样很清晰很多
我们可以把 dependencyManagement 放到单独的专门用来管理依赖的 POM 中，然后在需要使用依赖的模块中通过 import scope 依赖，就可以引入 dependencyManagement

1.我们可以写这样一个用于依赖管理的子模块 POM：

```
<project>
 <modelVersion>4.0.0</modelVersion>
 <groupId>wang.conge.demo</groupId>
 <artifactId>sample-dependency-infrastructure</artifactId>
 <packaging>pom</packaging>
 <version>1.0-SNAPSHOT</version>
 <dependencyManagement>
   <dependencies>
       <dependency>
         <groupId>junit</groupId>
         <artifactid>junit</artifactId>
         <version>4.8.2</version>
         <scope>test</scope>
       </dependency>
       <dependency>
         <groupId>log4j</groupId>
         <artifactid>log4j</artifactId>
         <version>1.2.16</version>
       </dependency>
   </dependencies>
 </dependencyManagement>
</project>
```

2. 然后我们的父模块只需要通过非继承的方式来引入这段依赖管理配置：

```
<dependencyManagement>
   <dependencies>
       <dependency>
         <groupId>wang.conge.demo</groupId>
         <artifactid>sample-dependency-infrastructure</artifactId>
         <version>1.0-SNAPSHOT</version>
         <type>pom</type>
         <scope>import</scope>
       </dependency>
   </dependencies>
```

```
    </dependencyManagement>
```

六．优化版本号

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-annotations</artifactId>
            <version>3.3.0.ga</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-commons-annotations</artifactId>
            <version>3.3.0.ga</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate</artifactId>
             <version>3.2.5.ga</version>
        </dependency>
    </dependencies>
```

优化成：

```
<properties>
    <hibernate.annotations.version>3.3.0.ga</hibernate.annotations.version>
</properties>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-annotations</artifactId>
            <version>${hibernate.annotations.version}</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-commons-annotations</artifactId>
            <version>${hibernate.annotations.version}</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate</artifactId>
            <version>${hibernate.annotations.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

最佳实践：
一组相关的组件一般都有相同的版本号，可以把相同的版本号抽取出来，统一定义。