DependEnCy Conflicts Assessment

Project description

DECCA is Maven plugin which detects dependency conflict issues between Java projects and third party libraries and assesses the issues' severity levels to warn developers whether the issues are benign or harmful (e.g., causing runtime exceptions).

Background

Software projects depend on an increasing number of third party libraries. Since a depended library might depend on other libraries, a host project would transitively depend on more libraries. Such intensive dependencies on third-party libraries can easily lead to dependency conflicts in practice. That is, multiple versions of the same library or class are presented on the classpath. When multiple classes with the same fully-qualified name exist in a Java project, the JVM will load one of them and shadow the others. If these classes are not compatible, the project can exhibit unexpected behaviors when it has components relying on the shadowed ones.

Maven does a good job in dependency conflict resolution, it usually applied the "nearest wins strategy" to choose the version that is nearer to the root (host project) of the dependency tree, or "first declaration wins strategy" to choose the first declared classes or libraries and "shadow" the ones with the same fully-qualified names or project coordinates. Consequently, it does not guarantee loading the most appropriate class. The dependency conflict issue arises when the loaded classes are not the expected ones of the project (i.e., the referenced feature set of the project is not fully covered by the loaded classes).

Maven can warn developers of duplicate JARs and classes, but they cannot identify whether the duplications are benign or harmful, which leads to developers may overlook the harmful ones and take no resolution actions.

Our goal

DECCA aims to detect dependency conflict issues and assess their severity levels according to their impacts on the system and maintenance costs. The severity levels are defined as follows:

Level 1: the feature set referenced by host project is a subset of the actual loaded feature set. Besides, the shadowed version completely cover the feature set used by the host project. This indicates that any orders of the specification of these duplicate classes on the classpath

will not induce serious runtime errors. Therefore, this is a benign conflict and will not affect the system reliability at runtime.

Level 2: the feature set referenced by host project is a subset of the actual loaded feature set. However, the shadowed feature set doesn't cover the referenced feature set. It is considered as a potential risk for system reliability since different orders of the specifications of these duplicate classes on the classpath (e.g., in different running environment or building platform) might induce runtime errors. Compared with warnings at Level 1, warnings at Level 2 needs more costs to maintain.

Level 3: It is a harmful conflict, as the actual loaded feature set does not consume the feature set referenced by host project. The runtime errors will occur when the expected feature cannot be accessed. However, in this case, the shadowed feature set completely cover the feature set referenced by host project. Therefore, it can be solved by adjusting the dependency order on the classpath, without changing any source code.

Level 4: It is a harmful conflict, as the actual loaded feature set does not cover the referenced feature set. Besides, the shadowed feature set does not consume the referenced feature set neither. Therefore, this type of conflicts can not be easily resolved by adjusting the dependency orders on the classpath. In this case, to solve these issues, it requires more efforts to ensure the multiple versions of classes could be referenced by host project.

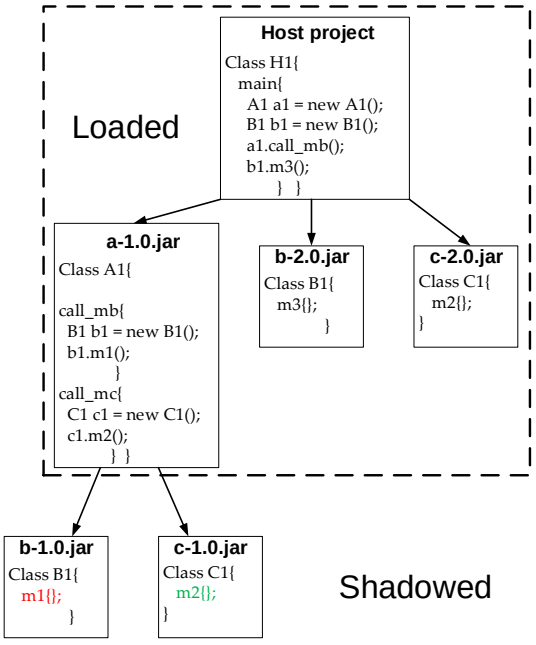

Figure 1 An illustrative example

A quick example

We give an illustrative example code to describe how DECCA generates issue reports. As shown in Fig.1, host project directly depends on **a-1.0.jar**, **b-2.0.jar** and **c-2.0.jar**, and libraries **b-1.0** and **c-1.0** are transitively introduced by **a-1.0.jar**. So that there are two

versions of libraries **b** and **c** in the system. According to Maven's nearest wins strategy, Maven chooses the version that appears at the nearest to the root (host project) of the dependency tree if there are multiple versions of the same library. In this case, only **b-2.0.jar** and **c-2.0.jar** can be loaded. By analyzing the dependencies, methods **m1()** (defined in library **b**) and **m2()** (defined in library **c**) are referenced by host project. Actually, **m3()** (defined in **b-2.0.jar**) and **m2()** (defined in **c-2.0.jar**) will be loaded, and **m1()** (defined in **b-1.0.jar**) and **m2()** (defined in **c-1.0.jar**) will be shadowed. For the dependency conflict issues, based on the above definition, DECCA assign the severity levels of library **b** (versions="2.0/1.0") and library **c** (versions="2.0/1.0") as **Level 4** and **Level 1**, respectively. The dependency issue report generated by Decca is described in Fig.2.

```
<?xml version="1.0" encoding="UTF-8"?>

<project project="neu.lab:decca.example.host" projectInfo="neu.lab:decca.example.host:1.0@D:\decca.example.host\pom.xml">
  <conflicts>
    <conflictJar groupId-artifactId="neu.lab:decca.example.b" versions="2.0/1.0" riskLevel="4">
      <versions>
        <version versionId="1.0" loaded="false">
          <path>neu.lab:decca.example.host:1.0:: + neu.lab:decca.example.a:1.0::compile + neu.lab:decca.example.b:1.0::compile</path>
        </version>
        <version versionId="2.0" loaded="true">
          <path>neu.lab:decca.example.host:1.0:: + neu.lab:decca.example.b:2.0::compile</path>
        </version>
      </versions>
      <RiskMethods tip="methods would be referenced but not be loaded!">
        <RiskMthd>neu.lab.decca.example.b.B1: void m1()</RiskMthd>
      </RiskMethods>
    </conflictJar>
    <conflictJar groupId-artifactId="neu.lab:decca.example.c" versions="2.0/1.0" riskLevel="1">
      <versions>
        <version versionId="2.0" loaded="true">
          <path>neu.lab:decca.example.host:1.0:: + neu.lab:decca.example.c:2.0::compile</path>
        </version>
        <version versionId="1.0" loaded="false">
          <path>neu.lab:decca.example.host:1.0:: + neu.lab:decca.example.a:1.0::compile + neu.lab:decca.example.c:1.0::compile</path>
        </version>
      </versions>
      <RiskMethods tip="methods would be referenced but not be loaded!"/>
    </conflictJar>
  </conflicts>
</project>
```

Figure 2 Dependency issue report generated by DECCA

License

DECCA is released under the MIT License.

Environment

Window operating system & JDK 1.7 or 1.8

Reference

This project references the open source projects **Soot**, **Apache Commons CSV**, **DOM4J**.

Contact author

wangying8052@163.com