【JAVA 基础】不同的 jar 拥有相同全限定类名和不同的方法 Method 时 NoSuchMethodError，同名类加载问题 / 双亲委派

同类名加载验证
前言
准备工程
demo-audi-1.0.jar 内容
demo-audi-2.0.jar 内容
demo-mercedes-1.0.jar 内容
开始演示功能
准备 Example 类
准备 Classpath
演示功能
自然顺序，*来代替所有 jar
手动改变 jar 名称，改变顺序
手动改变 jar 名称及版本，改变顺序
只保留一个 jar
手动指定 Classpath 的包先后顺序（ 重点）
结论
前言
最近工作中遇到了一个项目工程问题，在启动 jvm 的 classpath 有两个不同版本的 jwt 的 jar 包，在调用处报 java.lang.NoSuchMethodError: 其 Classpath 有两个不同版本的 jar 包，里面都有这个类，高版本的 jar 里面没有这个 Method，低版本有这个 Method。最终没有加载这个高版本的 Method
猜测此问题就是“全限定类名”完全一样，Cloassloader 只加载了高版本的 jar 包。

此文就是为了验证 jvm 是如何加载 classpath 中同类同全限定类名的过程。

准备工程
准备三个不同的 jar，里面都有同样一个类 Car，如下：

demo-audi-1.0.jar
demo-audi-2.0.jar
demo-mercedes-1.0.jar
这三个工程拥有同样一个 class: com.example.demo.Car

demo-audi-1.0.jar 内容

```
public class Car {

    private static final String version = "A4L";

    public String getVersion() {
        return version;
    }
```

```
    public String getName() {
        return "audi";
    }

    public Integer limitSpeed() {
        return 100;
    }

    public String seatPerson(Integer a) {
        return "可以坐 ： " + a;
    }
}
```

demo-audi-2.0.jar 内容

```
public class Car {

    private static final String version = "A6L";

    public String getVersion() {
        return version;
    }

    public String getName() {
        return "audi";
    }

    public Integer limitSpeed() {
        return 140;
    }

    public String seatPerson() {
        return "可以坐 ： 5";
    }
}
```

demo-mercedes-1.0.jar 内容

```
public class Car {

    private static final String version = "E300L";

    public String getVersion() {
        return version;
    }
```

```
    public String getName() {
        return "mercedes";
    }

    public Integer limitSpeed() {
        return 135;
    }

    public String hasPerson() {
        return "能舒服的坐 ： 4";
    }
}
```

开始演示功能
准备 Example 类
example.java

```
public class Example {

    public static void main(String[] args) {
        Car car = new Car();
        System.out.println("当前车辆版本： " + car.getVersion());
        System.out.println("当前 jar 包路径 ：  ");

System.out.println(car.getClass().getProtectionDomain().getCodeSource().getLocation().getP
ath());
        Method[] declaredMethods = car.getClass().getDeclaredMethods();
        for (Method declaredMethod : declaredMethods) {
            System.out.println("------------------");
            System.out.println("method name: " + declaredMethod.getName());
            List<String>                                collect                                =
Arrays.stream(declaredMethod.getParameterTypes()).map(Class::getName).collect(Collectors
.toList());
            if(!collect.isEmpty()) {
                System.out.println("parameter type : " + collect);
            }
            System.out.println("------------------");
        }
    }
}
```

准备 Classpath

test 目录

demo-audi-1.0.jar demo-audi-2.0.jar demo-example-1.0.jar demo-mercedes-1.0.jar
1
演示功能
自然顺序，*来代替所有 jar
java -classpath "/test/*" com.example.demo.Example

当前车辆版本：A4L
当前 jar 包路径 ：
/test/demo-audi-1.0.jar
----
method name: getName

----
method name: getVersion

----
method name: limitSpeed

----
method name: seatPerson
parameter type : [java.lang.Integer]

运行时，JVM 加载类 Car，根据【操作系统】的选择，本次加载了 demo-audi-1.0.jar 的 com.example.demo.Car 类的 class 文件

手动改变 jar 名称，改变顺序
当把 demo-audi-1.0.jar 重命名为 zemo-audi-1.0.jar ，“d”改成“z”

➜ test mv demo-audi-1.0.jar zemo-audi-1.0.jar
➜ test java -classpath "/test/*" com.example.demo.Example
当前车辆版本：A6L
当前 jar 包路径 ：
/test/demo-audi-2.0.jar
----
method name: getName

----
method name: limitSpeed

----
method name: getVersion

----
method name: seatPerson

运行时，JVM 加载类 Car，根据【操作系统】的选择，本次加载了 demo-audi-2.0.jar 的 com.example.demo.Car 类的 class 文件

手动改变 jar 名称及版本，改变顺序
当把 demo-audi-2.0.jar 重命名为 demo-mercedes-2.0.jar ，“audi”改成“mercedes”

[/test]# mv demo-audi-2.0.jar zemo-mercedes-2.0.jar
[/test]# ls demo-example-1.0.jar demo-mercedes-1.0.jar demo-mercedes-2.0.jar zemo-audi-1.0.jar
[/test]# java -classpath "/test/*" com.example.demo.Example
当前车辆版本：A6L
当前 jar 包路径 ：
/test/demo-mercedes-2.0.jar
----
method name: getName

----
method name: getVersion

----
method name: limitSpeed

----
method name: seatPerson

运行时，JVM 加载类 Car，根据【操作系统】的选择，本次加载了 demo-mercedes-2.0.jar 的 com.example.demo.Car 类的 class 文件

只保留一个 jar
如果把 audi 的两个 jar 移动出去，classpath 里面只剩下 demo-mercedes-1.0.jar 时

当前车辆版本：E300L
当前 jar 包路径 ：
/test/demo-mercedes-1.0.jar
----
method name: getName

----
method name: limitSpeed

```
------------------
method name: getVersion
------------------
------------------
method name: hasPerson
------------------
```

运行时，JVM 加载类 Car，根据【操作系统】的选择，本次加载了 demo-mercedes-1.0.jar 的 com.example.demo.Car 类的 class 文件

手动指定 Classpath 的包先后顺序（ 重点）

```
# 当 demo-audi-1.0.jar 在第一个时
java -classpath /test/demo-audi-1.0.jar:/test/demo-audi-2.0.jar:/test/demo-mercedes-1.0.jar:/test/demo-example-1.0.jar com.example.demo.Example

当前车辆版本：A4L
当前 jar 包路径 ：
/test/demo-audi-1.0.jar
------------------
method name: getName
------------------
------------------
method name: limitSpeed
------------------
------------------
method name: getVersion
------------------
------------------
method name: seatPerson
parameter type : [java.lang.Integer]

# 当 demo-audi-2.0.jar 在第一个时
java -classpath /test/demo-audi-2.0.jar:/test/demo-audi-1.0.jar:/test/demo-mercedes-1.0.jar:/test/demo-example-1.0.jar com.example.demo.Example

当前车辆版本：A6L
当前 jar 包路径 ：
/test/demo-audi-2.0.jar
------------------
method name: getName
------------------
------------------
method name: limitSpeed
------------------
```

```
------------------
method name: getVersion
------------------
------------------
method name: seatPerson
------------------

# 当 demo-mercedes-1.0.jar 在第一个时
java -classpath /test/demo-mercedes-1.0.jar:/test/demo-audi-2.0.jar:/test/demo-audi-1.0.jar:/test/demo-example-1.0.jar com.example.demo.Example
当前车辆版本：E300L
当前 jar 包路径 ：
/test/demo-mercedes-1.0.jar
------------------
method name: getName
------------------
------------------
method name: limitSpeed
------------------
------------------
method name: getVersion
------------------
------------------
method name: hasPerson
------------------
```

结论

根据 JVM 的双亲委派模型，默认情况下相同全限定类名的类只会加载一次，因此 JVM 加载 Car 类时只会从 demo-audi-1.0.jar 或 demo-audi-2.0.jar 以及 demo-mercedes-1.0.jar 选一个；

同名的两个 Car 类来自不同的三个 Jar 包，他们是平级的，根据 JVM 的类加载机制——双亲委派模型，相同全限定类名的类默认只会加载一次（除非手动破坏双亲委派模型）；

Jar 包中的类是使用 AppClassLoader 加载的，而类加载器中有一个命名空间的概念，同一个类加载器下，相同包名和类名的 class 只会被加载一次，如果已经加载过了，直接使用加载过的；

如果依赖中有多个全限定类名相同的类，那 JVM 会加载哪一个类呢？
比较靠谱的说法是，操作系统本身，控制了 Jar 包的默认加载顺序；也就是说，对于我们来说是不明确不确定的！

而 Jar 包的加载顺序，是跟 classpath 这个参数有关，当使用 idea 启动 springboot 的服务时，可以看到 classpath 参数的；包路径越靠前，越先被加载；

换句话说，如果靠前的 Jar 包里的类被加载了，后面 Jar 包里有同名同路径的类，就会被忽略掉，不会被加载

原文链接：https://blog.csdn.net/u013412066/article/details/129965950