由于对字节码修改功能的巨大需求，JDK 从 JDK5 版本开始引入了 `java.lang.instrument` 包。基本的思路是在 JVM 启动的时候添加一个代理，每个代理是一个 jar 包，其 MANIFEST.MF 文件里指定了代理类，这个代理类包含一个 premain 方法。JVM 在类加载时候会先执行代理类的 premain 方法，再执行 Java 程序本身的 main 方法，这就是 premain 名字的来源。在 premain 方法中可以对加载前的 class 文件进行修改。JDK6 还允许 JVM 在启动之后动态添加代理。

一句话来描述 javaagent 就是

> javaagent 是一个使用 instrumentation 的 API 用来改写类文件的 jar 包，可以看作是 JVM 的一个寄生插件。

0x01 javaagent 概览

javaagent 这个技术看起来非常神秘，很少有书会详细介绍。但是有很多工具是基于 javaagent 来实现的，比如热部署 JRebel、性能调试工具 XRebel、听云、newrelic 等。它能实现的基本功能包括

- 可以在加载 class 文件之前做拦截，对字节码做修改，可以实现 AOP、调试跟踪、日志记录、性能分析
- 可以在运行期对已加载类的字节码做变更，可以实现热部署等功能。

javaagent 概览如下图所示

0x02 一个小 demo

一个典型的 Agent 是这样的

```
public static class ClassLoggerTransformer
implements ClassFileTransformer {
        @Override
        public byte[] transform(ClassLoader
loader, String className, Class<?>
classBeingRedefined,
                                ProtectionDomain
protectionDomain, byte[] classfileBuffer) {
            System.out.println("transform: " +
className);
            Path path = Paths.get("/tmp/" +
className.replaceAll("/", ".") + ".class");
            try {
                Files.write(path,
classfileBuffer);
            } catch (IOException e) {
                e.printStackTrace();
            }
            return classfileBuffer;
        }
    }

    public static void premain(String
agentArgument, Instrumentation instrumentation) {
        ClassFileTransformer classFileTransformer
= new ClassLoggerTransformer();

instrumentation.addTransformer(classFileTransform
er, true);
    }
```

premain 有两个参数，第一个 agentArgument 是 agent 的启动参数，可以在 JVM 启动命令行中设置，比如`java -javaagent:<jarfile>=appId:agent-demo,agentType:singleJar test.jar`的情况下 agentArgument 的值为 "appId:agent-demo,agentType:singleJar"。第二个 instrumentation 是 java.lang.instrument.Instrumentation 的实例，可以通过 addTransformer 方法设置一个 ClassFileTransformer，这个 ClassFileTransformer 实现的 transform 方法能获取到类的名字和字节码。在上面的代码中，我们实现了打印类的名字和把类文件字节码 dump 到文件中的功能。

0x03 javaagent 如何打包

为了能够以 javaagent 的方式运行 premain 方法，我们需要将其打包成 jar 包，并在其中的 MANIFEST.MF 配置文件中，指定 Premain-class 等信息，一个典型的生成好的 MANIFEST.MF 内容如下

```
Premain-Class: me.geek01.javaagent.AgentMain
Agent-Class: me.geek01.javaagent.AgentMain
Can-Redefine-Classes: true
Can-Retransform-Classes: true
```

下面是一个可以帮助生成上面 MANIFEST.MF 的 maven 配置

```
<build>
  <finalName>my-javaagent</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifestEntries>
            <Agent-
Class>me.geek01.javaagent.AgentMain</Agent-Class>
            <Premain-
Class>me.geek01.javaagent.AgentMain</Premain-
Class>
            <Can-Redefine-Classes>true</Can-
Redefine-Classes>
            <Can-Retransform-Classes>true</Can-
Retransform-Classes>
          </manifestEntries>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

测试的类如下

```
public class Test01 {
    public static void main(String[] args) {
        System.out.println("in test01 main");
    }
}
javac Test01
java -javaagent:/path/to/my-javaagent.jar Test01
```

输出结果如下，可以看到一个最简单的 Main 函数会加载这些类。

```
transform: java/lang/invoke/MethodHandleImpl
transform: java/lang/invoke/MethodHandleImpl$1
transform: java/lang/invoke/MethodHandleImpl$2
transform: java/util/function/Function
transform: java/lang/invoke/MethodHandleImpl$3
transform: java/lang/invoke/MethodHandleImpl$4
transform: java/lang/ClassValue
transform: java/lang/ClassValue$Entry
transform: java/lang/ClassValue$Identity
transform: java/lang/ClassValue$Version
transform: java/lang/invoke/MemberName$Factory
transform: java/lang/invoke/MethodHandleStatics
transform: java/lang/invoke/MethodHandleStatics$1
transform: sun/misc/PostVMInitHook
transform: sun/usagetracker/UsageTrackerClient
transform:
java/util/concurrent/atomic/AtomicBoolean
transform: sun/usagetracker/UsageTrackerClient$1
transform: sun/usagetracker/UsageTrackerClient$4
transform: sun/usagetracker/UsageTrackerClient$3
transform: java/io/FileOutputStream$1
transform: sun/launcher/LauncherHelper
transform: sun/misc/URLClassPath$FileLoader$1
transform: Test01
transform: sun/launcher/LauncherHelper$FXHelper
transform: java/lang/Class$MethodArray
transform: java/lang/Void
in test01 main
transform: java/lang/Shutdown
transform: java/lang/Shutdown$Lock
```

0x04 agentmain

在 JDK5 中，开发者只能在 premain 中施展想象力，Instrumentation 也仅限于 main 函数执行前，这样的方式存在一定的局限性。
前面提到 premain 可以在类加载之前做字节码的修改，没有侵入性，对业务透明，但是在类加载以后就无能为力了，只能通过重新创建 ClassLoader 这种方式来重新加载。agentmain 为了解决这个问题应运而生。除了在命令行中指定 javaagent，现在可以通过 Attach API 远程加载。具体用法如下

```
public class AttachApiDemo {
    public static void main(String[] args) throws
Exception {
        if (args.length <= 1) {
            System.out.println("args size must >=
2, usage: java AttachApiDemo <pid>
/path/to/myagent.jar");
            return;
        }
        VirtualMachine vm =
VirtualMachine.attach(args[0]);
        vm.loadAgent(args[1]);
    }
}
```

使用 VirtualMachine.attach 加载的 agent 不会先于 main 函数执行，它是在 JVM 进程启动后再挂载上去的，它运行的入口不是 premain，而是 agentmain

```
public class AgentMain {
    public static void agentmain(String
agentArgs, Instrumentation inst) {
        System.out.println("agentmain called: " +
agentArgs);
        inst.addTransformer(new
ClassFileTransformer() {
            @Override
            public byte[] transform(ClassLoader
loader, String className, Class<?>
classBeingRedefined,

ProtectionDomain protectionDomain, byte[]
classfileBuffer) {
                System.out.println("agentmain
load Class  :" + className);
                return classfileBuffer;
            }
        }, true);
    }
}
```

待 attach 的 jvm 进程如下，使用 java MyAgentDemo运行，执行 jps 获取进程号

```
public class MyAgentDemo {
    public static void main(String[] args) throws
InterruptedException {
        for (int i = 0; i < 100000000; i++) {
            System.out.println("print: " + i);
            TimeUnit.MILLISECONDS.sleep(500);
        }
    }
}
```

执行java me.geek01.javaagent.AttachApiDemo <pid> /path/to/my-javaagent.jar就可以在 MyAgentDemo 进程中打印agentmain called: null，表示已经成功 attach 上去了。agentmain 函数也有一个 Instrumentation 参数，可以拦截类加载事件、对字节码做任意的修改。

0x05 小结

这篇文章讲解了 javaagent，一起来回顾一下要点：第一，javaagent 是一个使用 instrumentation 的
API 用来改写类文件的 jar 包，可以看作是 JVM 的一个寄生插件。第二，javaagent 有两个重要的入口类：Premain-Class 和 Agent-Class，分别对应入口函数 premain 和 agentmain，其中 agentmain 可以采用远程 attach API 的方式远程挂载另一个 JVM 进程。第三，介绍了 javaagent 的 maven 打包如何配置。

0x06 思考

留一道思考题，阿里最近开源了 Java 诊断利器 Arthas (https://github.com/alibaba/arthas)，你能分析它采用的是什么原理吗?

欢迎你在留言区留言，和我一起讨论。