S-javaagent

参考文章

javaagent使用指南

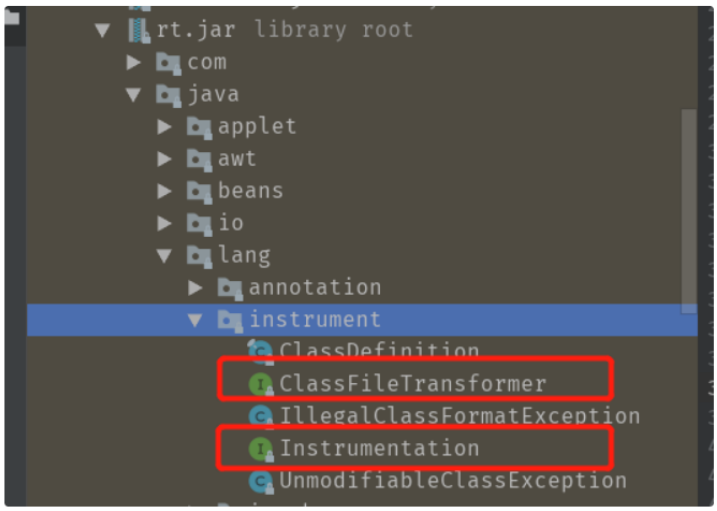
Java Instrumentation

javaagent实例

> Javaagent是java命令的一个参数。参数 javaagent 可以用于指定一个jar包，并且对该 java 包有2个要求：1、这个 jar 包的 MANIFEST.MF 文件必须指定 Premain-Class项。2、Premain-Class指定的那个类必须实现premain()方法。premain 方法，从字面上理解，就是运行在 main 函数之前的的类。当Java 虚拟机启动时，在执行 main 函数之前，JVM 会先运行-javaagent所指定 jar 包内 Premain-Class 这个类的 premain 方法 。在命令行输入 java 可以看到相应的参数，其中有 和 java agent相关的：

```
-agentlib:<libname>[=<选项>] 加载本机代理库 <libname>, 例如 -agentlib:hprof
   另请参阅 -agentlib:jdwp=help 和 -agentlib:hprof=help
-agentpath:<pathname>[=<选项>]
   按完整路径名加载本机代理库
-javaagent:<jarpath>[=<选项>]
   加载 Java 编程语言代理, 请参阅 java.lang.instrument
```

在上面 -javaagent 参数中提到了参阅 java.lang.instrument ，这是在 rt.jar 中定义的一个包，该路径下有两个重要的类：



该包提供了一些工具帮助开发人员在 Java 程序运行时，动态修改系统中的 Class 类型。其中，使用该软件包的一个关键组件就是 Javaagent。从名字上看，似乎是个 Java 代理之类的，而实际上，他的功能更像是一个Class 类型的转换器，他可以在运行时接受重新外部请求，对Class类型进行修改。

从本质上讲，Java Agent 是一个遵循一组严格约定的常规 Java 类。 上面说到 javaagent命令要求指定的类中必须要有premain()方法，并且对premain方法的签名也有要求，签名必须满足以下两种格式：

```
public static void premain(String agentArgs, Instrumentation inst)

public static void premain(String agentArgs)
```

JVM 会优先加载 带 Instrumentation 签名的方法，加载成功忽略第二种，如果第一种没有，则加载第二种方法。这个逻辑在

agent编写

- `MANIFREST.MF`

```
Manifest-Version: 1.0
Can-Redefine-Classes: true
Can-Retransform-Classes: true
Premain-Class: PreMainTraceAgent
//此处是一个空行，不可省略
```

- `PreMainTraceAgent`

```
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;
import java.lang.instrument.Instrumentation;
import java.security.ProtectionDomain;

public class PreMainTraceAgent {
    public static void premain(String agentArgs, Instrumentation inst){
        System.out.println("agentArgs : " + agentArgs);
        inst.addTransformer(new DefineTransformer(), true);
    }

    static class DefineTransformer implements ClassFileTransformer{

        @Override
        public byte[] transform(ClassLoader loader, String className, Class<?>
classBeingRedefined, ProtectionDomain protectionDomain, byte[] classfileBuffer)
throws IllegalClassFormatException {
            System.out.println("premain load Class:" + className);
            return classfileBuffer;
        }
    }
}
```

> 此处需要注意的是自动打包的过程中，自定义的`MANIFREST.MF`会被修改，所以需要手动替换，或者是用maven打包，并自定义打包插件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>javaagent</artifactId>
        <groupId>org.example</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>agentClass</artifactId>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```

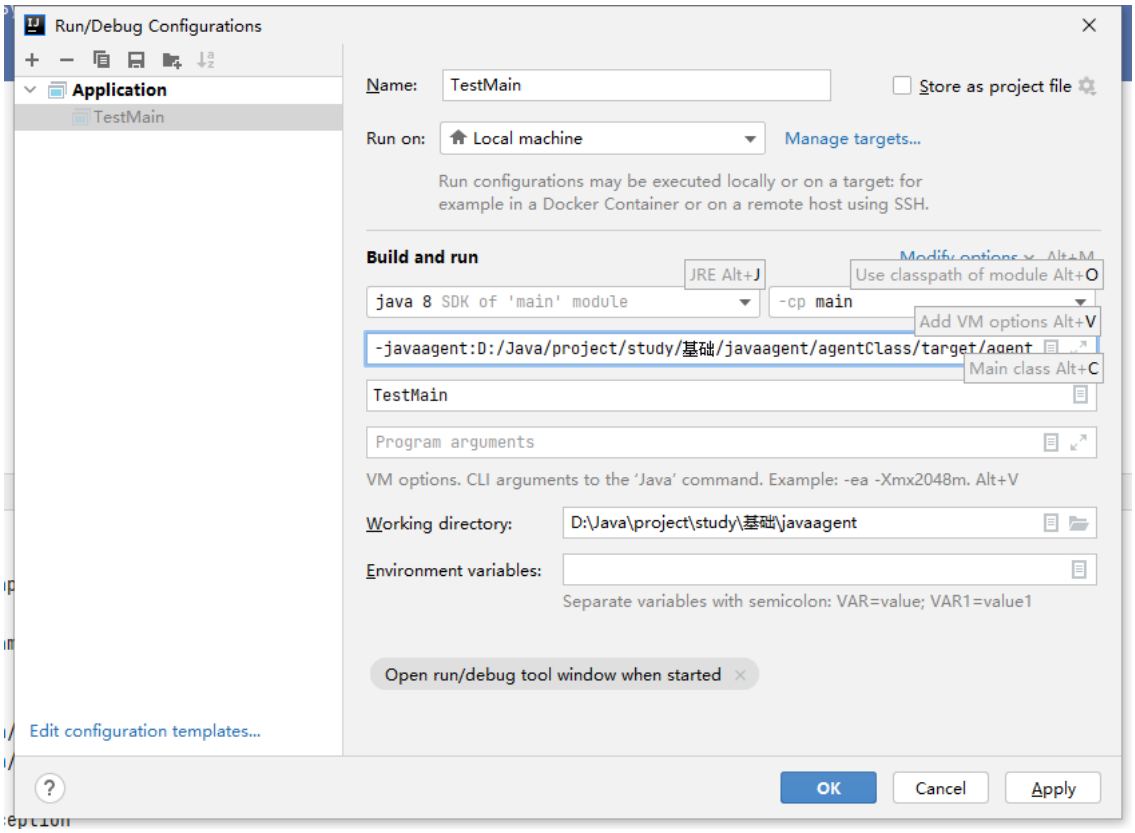
```
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.1.0</version>
                <configuration>
                    <archive>
                        <!--自动添加META-INF/MANIFEST.MF -->
                        <manifest>
                            <addClasspath>true</addClasspath>
                        </manifest>
                        <manifestEntries>
                            <Premain-Class>PreMainTraceAgent</Premain-Class>
                            <Agent-Class>PreMainTraceAgent</Agent-Class>
                            <Can-Redefine-Classes>true</Can-Redefine-Classes>
                            <Can-Retransform-Classes>true</Can-Retransform-
Classes>
                        </manifestEntries>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>
</project>
```

`main`函数编写

> 新建一个项目，创建main函数，在启动时通过`-javaagent`参数执行`premain`函数

```
public class TestMain {
    public static void main(String[] args) {
        System.out.println("main start");
        try {
            Thread.sleep(3000);
        }catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("main end");
    }
}
```

- 添加 `JVM` 参数，在启动时注入 `agent`



- 执行后会打印出全部的类加载过程

```
agentArgs : null
premain load Class:java/util/concurrent/ConcurrentHashMap$ForwardingNode
premain load Class:java/util/jar/Attributes
premain load Class:java/util/jar/Manifest$FastInputStream
premain load Class:java/util/jar/Attributes$Name
premain load Class:java/lang/Package
premain load Class:com/intellij/rt/execution/application/AppMainV2$Agent
premain load Class:com/intellij/rt/execution/application/AppMainV2
premain load Class:java/lang/NoSuchMethodException
premain load Class:java/lang/reflect/InvocationTargetException
premain load Class:com/intellij/rt/execution/application/AppMainV2$1
premain load Class:java/net/Socket
premain load Class:java/lang/invoke/MethodHandleImpl
premain load Class:java/net/InetSocketAddress
premain load Class:java/net/SocketAddress
```

动态修改字节码内容

- 运行 `main` 函数之前首先通过 `javassit` 修改 `date` 方法的类

```
import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtMethod;

import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;
import java.security.ProtectionDomain;

public class MyClassTransformer implements ClassFileTransformer {
    @Override
```

```
    public byte[] transform(ClassLoader loader, String className, Class<?>
classBeingRedefined, ProtectionDomain protectionDomain, byte[] classfileBuffer)
throws IllegalClassFormatException {
        // 操作Date类
        System.out.println(className);
        if ("java/util/Date".equals(className)) {
            System.out.println(11111111);
            try {
                //java.util.Date
                // 从ClassPool获得CtClass对象
                final ClassPool classPool = ClassPool.getDefault();
                final CtClass clazz = classPool.get("java.util.Date");
                CtMethod convertToAbbr =
clazz.getDeclaredMethod("convertToAbbr");
                //这里对 java.util.Date.convertToAbbr() 方法进行了改写，在 return之前
增加了一个 打印操作
                String methodBody = "
{sb.append(Character.toUpperCase(name.charAt(0)));" +
                        "sb.append(name.charAt(1)).append(name.charAt(2));" +
                        "System.out.println(\"sb.toString()\");" +
                        "return sb;}";
                convertToAbbr.setBody(methodBody);

                // 返回字节码，并且detachCtClass对象
                byte[] byteCode = clazz.toBytecode();
                //detach的意思是将内存中曾经被javassist加载过的Date对象移除，如果下次有需
要在内存中找不到会重新走javassist加载
                clazz.detach();
                return byteCode;
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
        // 如果返回null则字节码不会被修改
        return null;
    }
}
```

- 重新打包然后运行之后

```
sun/nio/cs/US_ASCII$Decoder
sun/misc/VMSupport
java/util/Hashtable$KeySet
sun/nio/cs/ISO_8859_1$Encoder
sun/nio/cs/Surrogate$Parser
sun/nio/cs/Surrogate
java/util/Date
11111111
sun/util/calendar/CalendarSystem
sun/util/calendar/Gregorian
sun/util/calendar/BaseCalendar
sun/util/calendar/AbstractCalendar
java/util/TimeZone
sun/util/calendar/ZoneInfo
```

JVM启动后动态Instrument

> 上面介绍的Instrumentation是在JDK 1.5中提供的，开发者只能在main加载之前添加手脚，在Java SE 6 的Instrumentation当中，提供了一个新的代理操作方法：agentmain，可以在main函数开始运行之后再运行。

- agentmain的编写

```
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;
import java.lang.instrument.Instrumentation;
import java.security.ProtectionDomain;

public class AgentMain {
    public static void agentmain(String agentArgs, Instrumentation
instrumentation) {
        instrumentation.addTransformer(new DefineTransformer(), true);
    }

    static class DefineTransformer implements ClassFileTransformer {

        @Override
        public byte[] transform(ClassLoader loader, String className, Class<?>
classBeingRedefined, ProtectionDomain protectionDomain, byte[] classfileBuffer)
throws IllegalClassFormatException {
            System.out.println("premain load Class:" + className);
            return classfileBuffer;
        }
    }
}
```

- 配置pom

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>javaagent</artifactId>
        <groupId>org.example</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>agentMain</artifactId>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.1.0</version>
                <configuration>
                    <archive>
                        <!--自动添加META-INF/MANIFEST.MF -->
                        <manifest>
```

```
                        <addClasspath>true</addClasspath>
                    </manifest>
                    <manifestEntries>
                        <Agent-Class>AgentMain</Agent-Class>
                        <Can-Redefine-Classes>true</Can-Redefine-Classes>
                        <Can-Retransform-Classes>true</Can-Retransform-
Classes>
                    </manifestEntries>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>

<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
</properties>

</project>
```

- `main`函数编写

> 通过`VirtualMachineDescriptor`获取所有运行中的`jvm`进程，然后匹配当前进程，最后加载`agent`

```
import com.sun.tools.attach.*;

import java.io.IOException;
import java.util.List;
public class TestMain2 {
    public static void main(String[] args) throws IOException,
AttachNotSupportedException {
        //获取当前系统中所有 运行中的 虚拟机
        System.out.println("running JVM start ");
        List<VirtualMachineDescriptor> list = VirtualMachine.list();
        for (VirtualMachineDescriptor vmd : list) {
            //如果虚拟机的名称为 xxx 则 该虚拟机为目标虚拟机，获取该虚拟机的 pid
            //然后加载 agent.jar 发送给该虚拟机
            System.out.println(vmd.displayName());
            if (vmd.displayName().endsWith("TestMain2")) {  //在获取到当前进程之后，
注入agent
                System.out.println("111111111111");
                VirtualMachine virtualMachine = null;
                try {
                    virtualMachine = VirtualMachine.attach(vmd.id());
                    virtualMachine.loadAgent("D:\\Java\\project\\study\\基础
\\javaagent\\agentMain\\target\\agentMain-1.0-SNAPSHOT.jar");
                    virtualMachine.detach();
                } catch (AgentLoadException e) {
                    e.printStackTrace();
                } catch (AgentInitializationException e) {
                    e.printStackTrace();
                }

            }
        }
```

```
    }
}
```

- 运行结果

```
running JVM start
org.jetbrains.jps.cmdline.Launcher D:/Program Files/JetBrains/IntelliJ IDEA 2021.1.1/lib/asm-all-9.1.jar;D:/Program Files/JetBrains/IntelliJ IDE
TestMain2
111111111111
premain load Class:java/lang/IndexOutOfBoundsException
org.jetbrains.idea.maven.server.RemoteMavenServer36

premain load Class:java/lang/Shutdown
premain load Class:java/lang/Shutdown$Lock
```