Java Instrumentation

在Java SE5以后，可以通过java.lang.instrument做动态Instrumentation，开发者可以使用Instrumentation构建一个独立于应用程序的代理程序（Agent），用来监测和协助运行在JVM上的程序，甚至可以替代和修改某些类的定义。通过该功能，开发者可以实现更为灵活的运行时虚拟机监控和Java类操作，这样的特性实际上提供一种虚拟机级别支持的AOP实现方式，使得开发者无需对JDK做任何升级和改动，就可以实现某些AOP功能。SE6以后，instrumentation包被赋予更强大的功能，启动后的instrument、本地代码（native code） instrumentation，以及动态改变classpath等，这些改变使得Java具有更强的动态控制、解释能力。

java.lang.instrument包的实现依赖于JVMTI（Java Virtual Machine Tool Interface），由Java虚拟机提供，为JVM相关的工具提供的本地编程接口集合，JVMTI提供了一套“代理”程序机制，可以支持第三方工具以代理的方式连接和访问JVM，并利用其提供的丰富的编程接口，完成很多JVM相关功能。在Instrumentation的实现中，存在一个JVMTI的代理程序，铜鼓调用JVMTI当中的Java类相关的函数来完成Java类的动态操作。

Instrumentation的代理执行方式有两种：通过-javaagent参数指定一个特定jar文件（包含Instrumentation代理）来启动代理程序；通过agentmain方式，在JVM启动后动态添加代理的方式。

# premain方式

在SE5当中，开发者可以让Instrumentation代理在main函数运行前执行，步骤如下：

1. 编写包含premain方法的代理程序

*public static void premain(String agentArgs, Instrumentation inst);  [1]*

*public static void premain(String agentArgs);[2]*

其中[1]的优先级比[2]高的，将会被优先执行，在这个premain函数中，开发者可以进行对类各种操作。

-agentArgs：premain函数的程序参数，随同-javaagent一起传入，与main函数不同的是整个参数是一个字符串而不是一个字符串数组，如果程序参数有多个，程序将自行解析这个字符串。

-inst:是一个java.lang.instrument.Instrumentation实例，由JVM自动传入，Instrumentation是instrument包中定义了一个接口，也是这个包的核心部分，集中其中几乎所有的功能方法，例如类定义的转换和操作等。

1. 将该类打包成Jar包

将这个Java类打包成一个jar文件，并在其中的manifest属性当中加入“Premain-Class”来指定步骤1当中编写那个带有premain的Java类。

1. 运行，用如下方式运行带有Instrumentation的Java程序，命令示例如下：

*java -javaagent:jar 文件的位置 [= 传入 premain 的参数 ]*

对Java类文件的操作，可以理解为对一个byte 的操作（将类文件的二进制字节流读入一个byte数组）。开发者可以在ClassFileTransfomer的transform方法中得到，操作并最终返回一个类定义（byte数组）。下面进行示例如下：

* 定义一个简单的类TransClass，通过一个静态方法返回这个整数：

*public class TransClass {*

*public int getNumber() {*

*return 1;*

*}*

*}*

* 编写测试类，TestMainInJar，定义如下：

*public class TestMainInJar {*

*public static void main(String[] args) {*

*System.out.println(new TransClass().getNumber());*

*}*

*}*

执行后可以得到输出“1”。然后，在TransClass的getNumber方法改成如下：

*public int getNumber() {*

*return 2;*

*}*

再将这个返回2的Java文件编译成类文件，然后将编写的类文件命名为TransClass.class.2。

* 然后建立一个Transfomer类：

*import java.io.File;*

*import java.io.FileInputStream;*

*import java.io.IOException;*

*import java.io.InputStream;*

*import java.lang.instrument.ClassFileTransformer;*

*import java.lang.instrument.IllegalClassFormatException;*

*import java.security.ProtectionDomain;*

*public class Transformer implements ClassFileTransformer {*

*public static final String classNumberReturns2 = "TransClass.class.2";*

*public static byte[] getBytesFromFile(String fileName) {*

*try {*

*File file = new File(fileName);*

*InputStream is = new FileInputStream(file);*

*long length = file.length();*

*byte[] bytes = new byte[(int) length];*

*int offset = 0;*

*int numRead = 0;*

*while (offset <bytes.length*

*&& (numRead = is.read(bytes, offset, bytes.length - offset)) >= 0) {*

*offset += numRead;*

*}*

*if (offset < bytes.length) {*

*throw new IOException("Could not completely read file "*

*+ file.getName());*

*}*

*is.close();*

*return bytes;*

*} catch (Exception e) {*

*System.out.println("error occurs in \_ClassTransformer!"*

*+ e.getClass().getName());*

*return null;*

*}*

*}*

*public byte[] transform(ClassLoader loader,*

*String className,*

*Class<?> classBeingRedefined,*

*ProtectionDomain protectionDomain,*

*byte[] classfileBuffer) throws IllegalClassFormatException {*

*if (!className.contains("TransClass")) {*

*return null;*

*}*

*return getBytesFromFile(classNumberReturns2);*

*}*

*}*

* 最后建立一个Premain类，写入Instrumentation的代理方法premain

*import java.lang.instrument.Instrumentation;*

*import java.lang.instrument.UnmodifiableClassException;*

*public class Premain {*

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

*throws ClassNotFoundException, UnmodifiableClassException {*

*inst.addTransformer(new Transformer());*

*}}*

addTransformer方法中并没有指明要转换哪个类，转换发生在premain函数执行之后，main函数执行之前，这个时候每装载一个类，transform方法就会执行一次，查看是否需要转换，所在在transform（Transfomer类）方法中，程序用className.equals(“TransClass”)来判断当前的类是否需要转换。

* 代码完成后，将其打包成jar，在manifest中加入如下属性来指定premain所在的类：

*Premain-Class：com.fys.Premain*

在pom.xml中添加plugin：

*<plugin>*

*<groupId>org.apache.maven.plugins</groupId>*

*<artifactId>maven-jar-plugin</artifactId>*

*<version>2.3.1</version>*

*<configuration>*

*<archive>*

*<manifest>*

*<addClasspath>true</addClasspath>*

*</manifest>*

*<manifestEntries>*

*<Premain-class>*

*com.fys.trans.Premain*

*</Premain-class>*

*</manifestEntries>*

*</archive>*

*</configuration>*

*</plugin>*

* 运行程序，用普通方式运行这个jar中的main函数，可以得到输出”1”，如果用下列方式运行：

*java -javaagent:aop-1.0-SNAPSHOT.jar -cp aop-1.0-SNAPSHOT.jar com.fys.test.TestMainInJar*

则会得到输出”2”。

除了用addTransformer的方式，Instrumentation当中还有另外一个方法redefineClasses来实现premain当中指定的转换，如下：

*public class Premain {*

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

*throws ClassNotFoundException， UnmodifiableClassException {*

*ClassDefinition def = new ClassDefinition(TransClass.class， Transformer*

*.getBytesFromFile(Transformer.classNumberReturns2));*

*inst.redefineClasses(new ClassDefinition[] { def });*

*System.out.println("success");*

*}*

*}*

# agentmain方式

在Java SE5基础上，Java SE6针对这种状况作出了改进，开发者可以在main函数开始执行之后，再启动自己的Instrumentation程序，定义一个跟premain并驾齐驱的agentmain方法，可以在main函数开始运行之后再运行。跟premain函数一样，开发者可以编写一个含有一个agentmain函数的Java类，如下：

*public static void agentmain (String agentArgs, Instrumentation inst);          [1]*

*public static void agentmain (String agentArgs);            [2]*

同样，[1]的优先级比[2]高，将会被优先执行，其方法参数与premain相同。与Premain-Class类似，开发者必须在manifest文件里面设置Agent-Class来指定agentmain函数的类。

在Java SE6当中提供Attach API，可以向目标JVM Attach代理工具程序，开发者可以通过外加的代理程序方便的监控JVM。Attach API，提供Attach动作和Detach动作（从JVM上面解除一个代理）等。

1. 编写测试类，Attach API写在一个线程里面，用睡眠等待的方式，每隔一秒时间检查一次所有的Java虚拟机，当发现新的虚拟机出现的时候，就调用attach函数，随后再按照Attach API文档中所说的方式装载Jar文件。

*public class TestAgentMainInJar {*

*public static void main(String[] args) throws InterruptedException {*

*System.out.println(new TransClass().getNumber());*

*int count = 0;*

*while(true) {*

*Thread.sleep(1000);*

*count++;*

*int number = new TransClass().getNumber();*

*System.out.println("Number: "+number);*

*}*

*}}*

程序运行后，输出Number:i

1. 编写AgentAfterMain类，实现agentmain方法，如下所示：

*import java.lang.instrument.Instrumentation;*

*public class AgentAfterMain {*

*public static void agentmain(String args, Instrumentation inst) {*

*System.out.println("LoadAgent After main run.args=" +args);*

*Class<?>[] classes = inst.getAllLoadedClasses();*

*for (Class<?> cls: classes) {*

*System.out.println(cls.getName());*

*}*

*System.out.println("Agent run completely");*

*}}*

输出加载的类。

1. 编写好Agent Jar后，通过Attach Tools API，使用com.sum.tools.attach包下的VirtualMachine类，使用loadAgent方法指定AgentMain类并加载，实现类：

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

*public class RunAttach*

*{*

*public static void main(String[] args) throws Exception*

*{*

*// args[0]传入的是某个jvm进程的pid*

*String targetPid = args[0];*

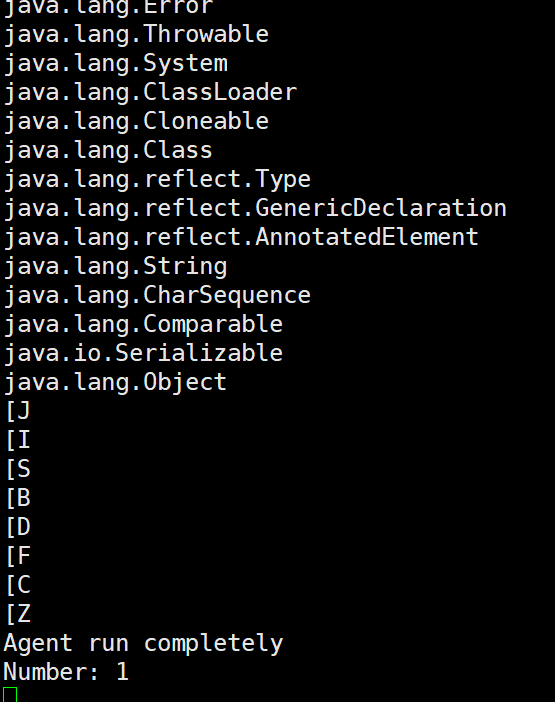
*VirtualMachine vm = VirtualMachine.attach(targetPid);*

*vm.loadAgent("~/agentmain.jar",  "toagent");*

*}*

*}*

执行后在*TestAgentMainInJar*的输出信息如下：



https://www.ibm.com/developerworks/cn/java/j-lo-jse61/index.html