*摘要: instrumentation 功能介绍（javaagent)*

    Instrumentation JDK中对它介绍如下：这个类为JVM上运行时的程序提供测量手段。很多工具通过Instrumenation 修改方法字节码 实现收集数据目的。这些通过Instrumentaion搜集数据的工具不会改变程序的状态和行为。这些良好的工具包括  monitoring agents  , ,profilers, coverage analyzers, 和 event loggers。

    有两种方式来获取Instrumentation接口实例：

* 启动JVM时指定agent类。这种方式，Instrumentation的实例通过agent class的premain方法被传入。
* JVM提供一种当JVM启动完成后开启agent机制。这种情况下，Instrumention实例通过agent代码中的的agentmain传入。

    java agent 在JDK package specification中解释：一个agent 是被作为Jar 文件形式来部署的。在Jar文件中manifest中指定哪个类作为agent类。具体的实现包括

通过命令行直接指定选项开启agent,也支持JVM启动程序后，通过工具attach到该程序上。

    下面通过例子来说明javaagent + Instrumentation的用法。

**通过在程序启动前  preagent方式**：（该例子实现输出所有JVM加载类名字，并在People类的 sayHello 方法调用前后加入log）

   1 people类

public class People {

    public void sayHello(){

        System.out.println("hello !!!!");

    }

}

    2 实现一个 ClassFileTransformer类：

        agent通过该具体实现来实现转换加载到JVM中class files。这种类的转换发生在类文件被载入JVM之前。因此这可以实现类AOP编程的效果。

public class PeopleClassFileTransformer implements ClassFileTransformer {

    /\*\*

     \* 通过javassist修改字节码

     \* @param loader

     \* @param className

     \* @param classBeingRedefined

     \* @param protectionDomain

     \* @param classfileBuffer

     \* @return

     \* @throws IllegalClassFormatException

     \*/

    @Override

    public byte[] transform(ClassLoader loader, String className, Class<?> classBeingRedefined, ProtectionDomain protectionDomain, byte[] classfileBuffer) throws IllegalClassFormatException {

        System.out.println("load class:"+className);

        if("com.yao.intrumentation.People".equals(className)){

            try {

                //通过javassist修改sayHello方法字节码

                CtClass ctClass= ClassPool.getDefault().get(className.replace('/','.'));

                CtMethod sayHelloMethod=ctClass.getDeclaredMethod("sayHello");

                sayHelloMethod.insertBefore("System.out.println(\"before sayHello----\");");

                sayHelloMethod.insertAfter("System.out.println(\"after sayHello----\");");

                return ctClass.toBytecode();

            } catch (NotFoundException e) {

                e.printStackTrace();

            } catch (CannotCompileException e) {

                e.printStackTrace();

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

        return classfileBuffer;

    }

}

  3 编写agent,该类必须包含premain方法。并在META-INF 中添加MANIFEST.MF ，在清单中添加

Premain-Class: com.yao.intrumentation.MyAgent

public class MyAgent {

    /\*\*

     \* 该方法是一个类作为agent类必备的

     \* @param agentArgs

     \* @param inst

     \*/

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

        //加入ClassFileTransfomer

        inst.addTransformer(new PeopleClassFileTransformer());

    }

}

  4 打包agent类（这里可以把上面的 ClassFileTransfer MyAgent单独拿出来打包 。这里为了方面把所有的代码都放到一起了。。）

    代码编译后  在target/classes/下打包   加 m 参数是指定MANIFEST

jar -cvfm myagent.jar META-INF/MANIFEST.MF \*         // 在自己项目目录下执行 比如maven目录结构

                                                     // 在编译后的target/class/下执行

 5 测试main类：

public class TestMain {

    public static void main(String[]args){

        People people=new People();

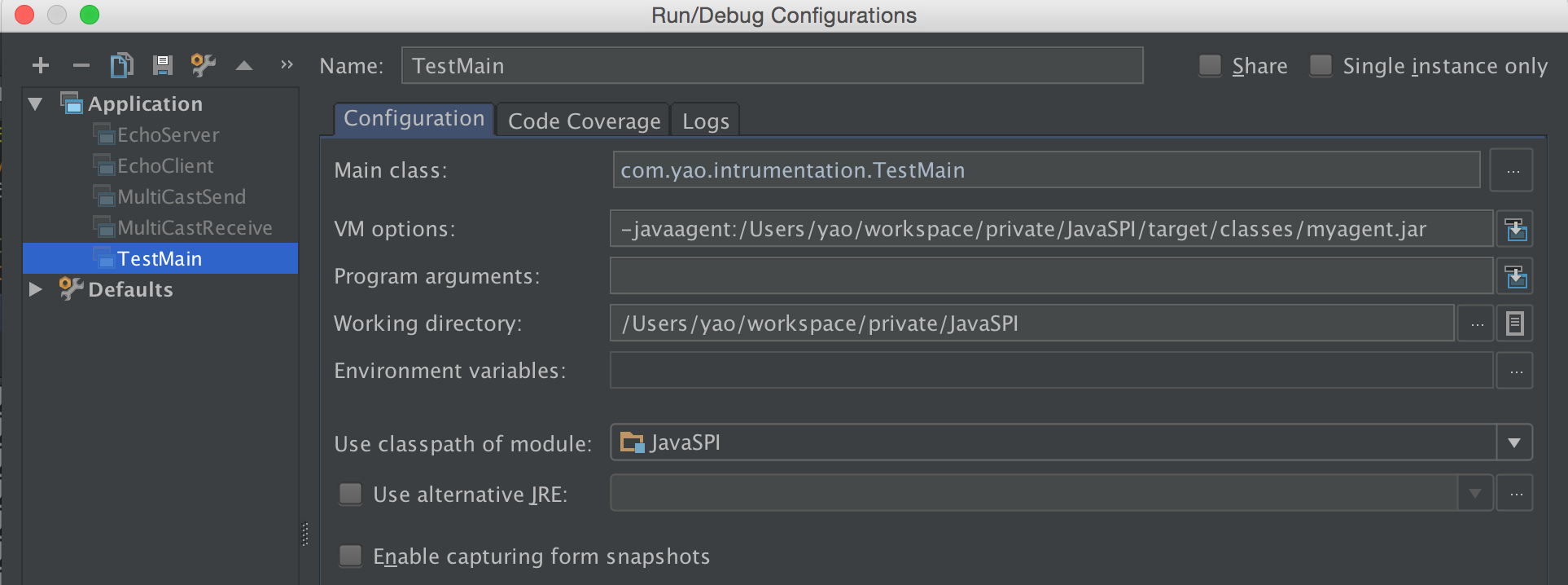
        people.sayHello();

    }

}

  启动 这里为了方便解决引用的javassist jar包 classpath问题，我直接在Intellij 指定VM参数启动上面的main 方法，这样就不用在命令行里手工设定classpath。

-javaagent:/Users/yao/workspace/private/JavaSPI/target/classes/myagent.jar 指代我打包的agent jar 位置。



结果输入如下：

load class:java/lang/invoke/MethodHandleImpl

load class:java/lang/invoke/MethodHandleImpl$1

load class:java/lang/invoke/MethodHandleImpl$2

load class:java/util/function/Function

load class:java/lang/invoke/MethodHandleImpl$3

load class:java/lang/invoke/MethodHandleImpl$4

load class:java/lang/ClassValue

load class:java/lang/ClassValue$Entry

load class:java/lang/ClassValue$Identity

load class:java/lang/ClassValue$Version

load class:java/lang/invoke/MemberName$Factory

load class:java/lang/invoke/MethodHandleStatics

load class:java/lang/invoke/MethodHandleStatics$1

load class:sun/misc/PostVMInitHook

load class:sun/launcher/LauncherHelper

load class:com/yao/intrumentation/TestMain

load class:sun/launcher/LauncherHelper$FXHelper

load class:java/lang/Class$MethodArray

load class:java/lang/Void

load class:com/yao/intrumentation/People

before sayHello----

hello !!!!

after sayHello----

load class:java/lang/Shutdown

load class:java/lang/Shutdown$Lock

   下面简单介绍通过attach到正在**运行的JVM程序的 agentmain方式**：

   1 编写agent类

public class MainAgent {

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

        Class[] classes = inst.getAllLoadedClasses();

        for(Class cls :classes){

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

        }

    }

}

  2 写一个长时间运行main

public class RunningApp {

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

        People people=new People();

        Thread.sleep(1000\*1000);

    }

}

  3 修改MANIFEST.MF

Agent-Class: com.yao.intrumentation.MainAgent

  用类似上面的方法打包成jar

  4 编写attach 程序

public class TestMainAgent {

    public static void main(String[]args) throws InterruptedException, IOException, AgentLoadException, AgentInitializationException, AttachNotSupportedException {

        VirtualMachine vm = VirtualMachine.attach(args[0]); //正在运行的java 程序 ps id

        vm.loadAgent("/Users/yao/workspace/private/JavaSPI/target/classes/agentmain.jar");

        //刚刚编译好的agent jar 位置

    }

}

运行 把 RunningApp启动后  jps 拿到ps id ,传给上面的程序，运行即可看到JVM加载的所有类文件

转载注明：<http://my.oschina.net/robinyao/blog/489767>

具体代码：<https://github.com/WangErXiao/JavaSPI/tree/master/src/main/java/com/yao/intrumentation>