jmtrace实验报告

盛朱恒 MF20330066

框架：使用**java.lang.Instrument**，劫持类的加载，插入字节码进行插桩。

1、JmtraceAgent类

创建JmtraceAgent类，并编写premain函数。在premain函数中，对**Instrumentation**添加**ClassFileTransformer**。重定义**ClassFileTransformer**类的**transform**方法，使其有以下功能。

首先，判断当前的类是否为java的库函数。

如果是库函数，那么保持原有的流程，不做任何操作，即直接返回**classfileBuffer**。

如果不是库函数，那么则需要进行操作，进行插桩，即使用**JmtraceClassAdapter**类进行改写。

2、JmtraceClassAdapter类

该类继承自**ClassVisitor**

重写了其**visitMethod**方法，使用**JmtraceMethodAdapter**在方法加载时进行真正的改写插桩。

3、JmtraceMethodAdapter类

该类继承自**MethodVisitor**，改写插桩的重点部分在该类中完成。

重写了两个方法，**visitInsn**与**visitFieldInsn**。访问内存的指令有： **getstatic，putstatic，getfield，putfield，*aload, *astore。**

其中，***aload, *astore**通过**visitInsn**访问，**getstatic，putstatic，getfield，putfield**通过**visitFieldInsn**访问。

重写visitInsn：

此处有个特点，java在对非数组的对象操作时，总会使用局部变量，在对数组的对象操作时，才会使用栈，也就是说，使用*aload/*store的对象，都是数组。

首先判断是load指令还是store指令。

load指令：load指令执行前，栈顶保留着此次load的对象及index，因此，将其复制一份压栈，并调用新增的**MyPrint**类中的对应函数，解析指定项，输出信息即可。

store指令：load指令执行前，栈顶保留着此次store的值、对象及index，新增的**MyPrint**类中的对应函数需要对象及index，因此需要将其复制后，调用新增的函数。这里需要注意，store的值有两种不同的长度，对double和long long, 会占用两层的栈，操作与其他情况不同。

总之，在load/store指令前，修改栈状态，为调用新增的对应函数准备参数，并使得调用完成后恢复原样。

最后，不论是什么指令，都要进行**super.visitInsn(opcode);**以保持其原有功能。

重写visitFieldInsn

这里有四个指令 **getstatic，putstatic，getfield，putfield**，这些指令是有操作数的。

getstatic/putstatic的相关信息都在操作数中，所以将操作数进行压栈，并调用**MyPrint**类中的对应函数。

getfield，putfield略微复杂，栈顶仍然有对象信息，因此需要复制栈顶，并将操作数压栈，最后调用**MyPrint**类中的对应函数。同时，**putfield**中也会有数据长度的问题，因此需要分类讨论。

总之，在指令执行前，修改栈状态，为调用新增的对应函数准备参数，并使得调用完成后恢复原样。

最后，不论是什么指令，都要进行**super.visitFieldInsn(opcode, owner, name, descriptor);**以保持其原有功能。

4、MyPrint类

该类是添加插桩时使用的输出函数。主要任务是解析前面传递的参数信息，并进行输出。

5、遇到的挑战

1、学习正确使用**java.lang.Instrument**以及学习java字节码

2、开始时，未发现java在处理单个的对象时，会使用局部变量，而非栈，因此不会使用到***aload/*store**指令，而是其他的**load/store**指令，导致使用测试程序时出现没有输出的情况。

3、开始时，未考虑到不同数据类型的长度问题，导致出现时有时无的、难以查找的bug。