Java Bytecode instrumentation

Modify your compiled code!

Paweł Oczadly

Bytecode instrumentation - what is it?

- Can modify compiled Java code! (bytecode)
- Exists since JDK 1.5
- Is achieved by using **Java agent**
  - **Can be run statically**

```
java -javaagent:Agent.jar=arguments_string Instrumented.jar
```

  - **Or be attached into the running JVM (Java code)**

```
VirtualMachine vm = VirtualMachine.attach(processId);
vm.loadAgent(agentJarPath);
vm.detach();
```

Okay… but what is the bytecode?

Java code (part of *SamplePrinter.java*)

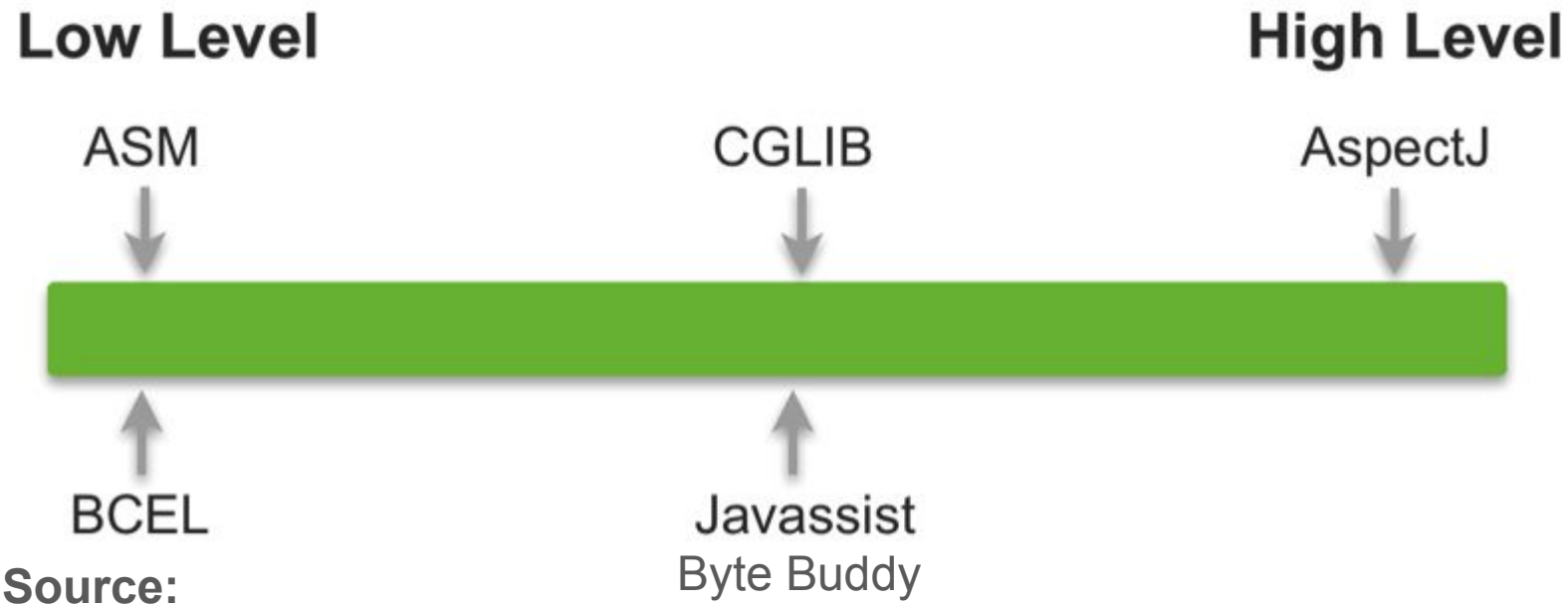
```
public void print1() {
    logger.info("Hello guys");
}
```

```
javap -c SamplePrinter.class
```

Bytecode of print1 class (part of *SamplePrinter.class*)

```
public void print1();
        Code:
        0: getstatic #2                  // Field logger:Lorg/slf4j/Logger;
        3: ldc       #3                  // String Hello guys
        5: invokeinterface #4,  2  // InterfaceMethod org/slf4j/Logger.info...
        10: return
```

What tools and libraries can I use?



Source:
https://blog.newrelic.com/engineering/diving-bytecode-manipulation-creating-audit-log-asm-javassist/

Why should I bother myself with it?

- Adding functionality to existing code (we don’t have access to the original project)
- Adding ad-hoc patch
- Profiling
- Testing memory consumption
- Implementing breakpoint in debugger
- Checking coverage of tests
- Lazy load data from databases using proxies

How static agent should look like?

Java programs have:

```
public static void main()
```

Or

```
public static void main(String[] args)
```

Java agents have:

```
public static void premain(String agentArgs)
```

Or

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

Is passed by the classloader. We can register our transfomers with use of this instance.

What is transformer?

- Class that implements **ClassFileTransformer** interface
- Every transformer is invoked when classloader loads a new class
- It can manipulate a bytecode of a loaded class

```
public byte[] transform(ClassLoader loader, String className,
        Class classBeingRedefined, ProtectionDomain protectionDomain,
                byte[] classfileBuffer) throws IllegalClassFormatException
```

How to build static agent with Gradle?

```
jar {
   manifest {
      attributes(
            'Premain-Class': 'InstrumentationAgent',
            'Can-Redefine-Classes': 'true',
            'Can-Retransform-Classes': 'true',
            'Can-Set-Native-Method-Prefix': 'true',
            'Implementation-Title': "ClassLogger",
            'Implementation-Version': rootProject.version
      )
   }
}
```

Let’s log methods’ execution time!

We have a simple program:

```
public static void main(String[] args)
 throws InterruptedException {
   logger.info("Starting");
   delay(2000);
   samplePrinter.print1();
   samplePrinter.print2();
   delay(2000);
   advancedPrinter.printArgument("My amazing
argument");
}
```

Commands

```
./script1.sh run
./script1.sh run_with_example_args
```

To put it simple - our transformer method should look like this:

```
try {
      ClassPool classPool = ClassPool.getDefault();
      CtClass ctClass = classPool.makeClass(new
ByteArrayInputStream(classfileBuffer));
      CtMethod[] methods = ctClass.getDeclaredMethods();
      for (CtMethod method : methods) {
         method.addLocalVariable("startTime", CtClass.longType);
        method.insertBefore(String.format("logger.debug(\"%s\);",
                                  "My instrumented message"));
      }
      byteCode = ctClass.toBytecode();
      ctClass.detach();
   } catch (Exception exception) {
      ...
   }
}
return byteCode;
```

Let’s instrument classes based on annotations

Our annotation definition:

```
public @interface DetailedLogs {}
```

Most important part of our instrumentation

```
public static final String ANNOTATION_TYPE_NAME =
"part2.DetailedLogs";
```

```
ClassPool classPool = ClassPool.getDefault();
// we can do it also this way
CtClass ctClass = classPool.get(className.replace("/", "."));

if (ctClass.hasAnnotation(ANNOTATION_TYPE_NAME)) {
   byteCode = instrumentClass(ctClass, className);
}
ctClass.detach();
```

Commands

```
./script2.sh run
./script2.sh run_instrumented
```

Attach to running Spring application

We are going to use the **Attach API** to the agent to the running JVM!

To do that we need to retrieve PID of the target JVM

```
jps | grep $APP_NAME | awk '{print $1}'
```

Code used to load the agent dynamically:

```
VirtualMachine vm = VirtualMachine.attach(processId);
vm.loadAgent(agentJarPath, agentArguments);
vm.detach();
```

We need to add the following dependency to our agent loader

```
dependencies {
files("${System.getProperty('java.home')}/../lib/tools.jar")
}
```

Dynamic agent differences

build.gradle

```
jar {
  manifest {
    attributes(
        'Agent-Class' : "InstrumentationAgent",
        'Can-Redefine-Classes': 'true',
        'Can-Retransform-Classes': 'true',
        'Can-Set-Native-Method-Prefix': 'true',
        'Implementation-Title': "ClassLogger",
        'Implementation-Version': rootProject.version
    )
  }
}
```

We need to find already loaded class and its class loader (Spring Boot uses its own class loader)

```
    for(Class<?> clazz: instrumentation.getAllLoadedClasses()) {
        if (isClassNameMatching(classNamesToInstrument, clazz)) {
            ClassLoader cLoader = clazz.getClassLoader();
            ClassFileTransformer transformer =
createTransformer(cLoader);
            performInstrumentation(instrumentation, clazz, transformer);
        }
    }
```

What is different in transforming loaded classes?

- Bytecode is already loaded into the memory, therefore we need to retransform our classes
- We must explicitly specify that our transformer can retransform classes
- Right after retransforming we are going to remove the transformer

Commands

```
./script3.sh run
./script3.sh
attach_agent_with_example_args
```

```
instrumentation.addTransformer(classFileTransformer, true);

try {
   instrumentation.retransformClasses(clazz);
} catch (Exception ex) {
   throw new RuntimeException("Failed to transform [" + clazz.getName() + "]", ex);
} finally {
   instrumentation.removeTransformer(classFileTransformer);
}
```

Sources

- https://javapapers.com/core-java/java-instrumentation/

- **https://medium.com/@jakubhal/instrumentation-of-spring-boot-application-with-byte-buddy-bbd28619b7c**
- https://web.archive.org/web/20141014195801/http://dhruba.name/2010/02/07/creation-dynamic-loading-and-instrumentation-with-javaagents/
- https://www.infoq.com/articles/Living-Matrix-Bytecode-Manipulation/
- https://blog.newrelic.com/engineering/diving-bytecode-manipulation-creating-audit-log-asm-javassist/

- **https://stackoverflow.com/questions/18567552/how-to-retransform-a-class-at-runtime**

- **https://www.javassist.org/tutorial/tutorial.html**