BYTECODE WEAVING 101

Michael Rozumyanskiy

The Story of
DEPENDENCY INJECTION

We started our project in Kotlin and chose Guice as a DI framework. Later we found out that it was working too slow. One possible solution was to switch to Dagger. But at that time Kotlin didn't have support for APT. We had to make a tough decision: to give up DI or to come up with a more effective solution.

Being true Jedi, we decided to develop our own lightning-fast DI framework and called it Lightsaber. In order to make it work at a compile time we had to join the Dark Side and modify bytecode. And here we are...

USERS

InstantRun ProGuard DexGuard JRebel

JaCoCo EasyMock Mockito PowerMock jMock Realm

Retrolambda hugo frodo redex

WHY?

- Persistence
- Dependency Injection
- Mocking
- Security
- Tools
- Analysis

BYTECODE

“Java bytecode is the instruction set of the Java virtual machine.”

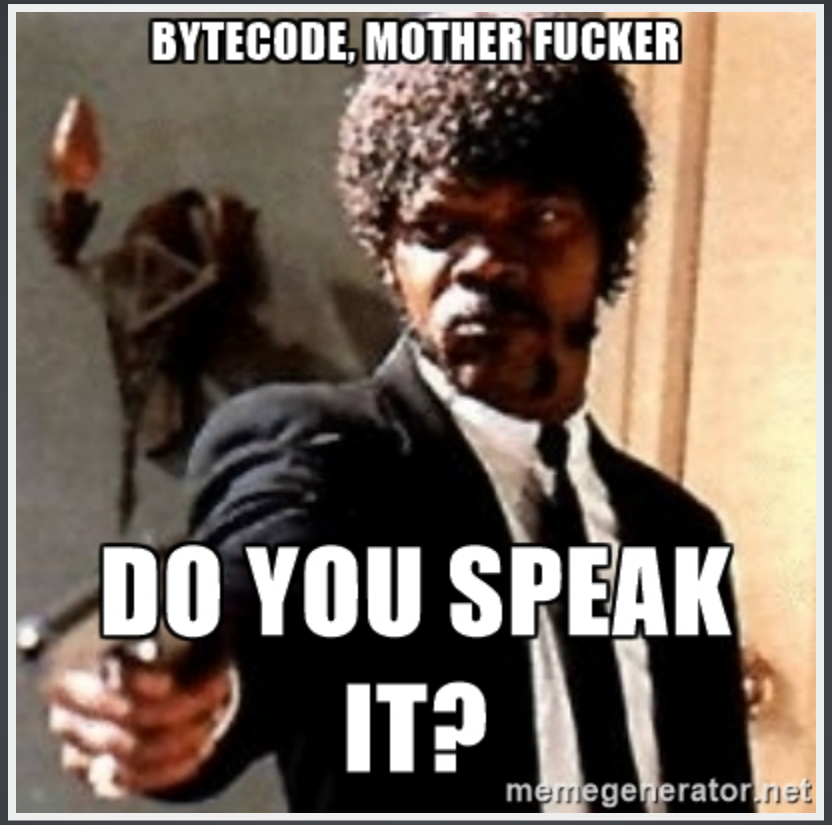
THIS JAVA CODE

```
private void checkErrors() {
  if (errorReporter.hasErrors()) {
    throw new ProcessingException(composeErrorMessage());
  }
}
```

COMPILES TO...

5.3

```
// access flags 0x12
private final checkErrors()V
 L0
  LINENUMBER 101 L0
  ALOAD 0
  GETFIELD io/michaelrocks/lightsaber/processor/ClassProcessor.errorReporter :
    Lio/michaelrocks/lightsaber/processor/ErrorReporter;
  INVOKEVIRTUAL io/michaelrocks/lightsaber/processor/ErrorReporter.hasErrors ()Z
  IFEQ L1
 L2
  LINENUMBER 102 L2
  NEW io/michaelrocks/lightsaber/processor/ProcessingException
  DUP
  ALOAD 0
  INVOKESPECIAL io/michaelrocks/lightsaber/processor/ClassProcessor.composeErrorMessage
    ()Ljava/lang/String;
  INVOKESPECIAL io/michaelrocks/lightsaber/processor/ProcessingException.<init>
    (Ljava/lang/String;)V
  CHECKCAST java/lang/Throwable
  ATHROW
 L1
  LINENUMBER 104 L1
 FRAME SAME
  RETURN
 L3
  LOCALVARIABLE this Lio/michaelrocks/lightsaber/processor/ClassProcessor; L0 L3 0
  MAXSTACK = 3
  MAXLOCALS = 1
```


BYTECODE, MOTHER FUCKER
DO YOU SPEAK IT?
memegenerator.net

SOMETIMES YOU HAVE TO

6.2

```
Exception in thread "main" java.lang.VerifyError:
Bad type on operand stack in putfield
Exception Details:
  Location:
    io/michaelrocks/lightsaber/sample/LightsaberSample$MembersInjector
    .injectFields(Lio/michaelrocks/lightsaber/Injector;Ljava/lang/Object;)V @24: putfield
  Reason:
    Type 'io/michaelrocks/lightsaber/Injector' (current frame, stack[1])
    is not assignable to 'io/michaelrocks/lightsaber/sample/LightsaberSample' (constant pool 38)
  Current Frame:
    bci: @24
    flags: { }
    locals: {
      'io/michaelrocks/lightsaber/sample/LightsaberSample$MembersInjector',
      'io/michaelrocks/lightsaber/Injector',
      'java/lang/Object',
      'io/michaelrocks/lightsaber/Injector'
    }
    stack: {
      'io/michaelrocks/lightsaber/sample/LightsaberSample',
      'io/michaelrocks/lightsaber/Injector'
    }
```

LIBRARIES

- ASM
- Java Compiler API
- Grip

ASM

*"**ASM** is an all purpose Java bytecode manipulation and analysis framework."*

ANALYSIS & MANIPULATION


```
public class MyClassVisitor extends ClassVisitor {
  public MyClassVisitor(ClassVisitor delegate) {
    super(ASM5, delegate);
  }

  public void visit(int version, int access, String name,
      String signature, String superName, String[] interfaces) {
    int newAccess = access & ~ACC_FINAL;
    super.visit(version, newAccess, name, signature superName,
        interfaces);
  }
}
```

GENERATION


```
class ProviderGenerator extends GeneratorAdapter {
  void newProvider(Provider provider) {
    newInstance(provider.type);
    dup();
    loadArg(0);
    Method constructor =
        new Method("<init>", Type.VOID_TYPE, INJECTOR_TYPE);
    invokeConstructor(provider.type, constructor);
  }
}
```

JAVA COMPILER API

JSR-199

MORE READABLE...


```
appendln("package $packageName;")
appendln()
appendln("public class $className {")
appendln("  public static String ${DEOBFUSCATION_METHOD.name}(final int
appendln("    final short[] stringIndexes = indexes[id];")
appendln("    final char[] stringChars = new char[stringIndexes.length
appendln("    for (int i = 0; i < stringIndexes.length; ++i) {")
appendln("      stringChars[i] = chars[stringIndexes[i]];")
appendln("    }")
appendln("    return new String(stringChars);")
appendln("  }")
// Some other stuff...
appendln("}")
```

... THAN ASM


```
ClassWriter cw = new ClassWriter(COMPUTE_MAXS | COMPUTE_FRAMES);
cw.visit(V1_6, ACC_PUBLIC | ACC_SUPER, name, null,
    "java/lang/Object", null);
MethodVisitor mv = cw.visitMethod(ACC_PUBLIC | ACC_STATIC,
    DEOBFUSCATION_METHOD.name, "(I)Ljava/lang/String;", null, null);
mv.visitCode();
mv.visitFieldInsn(GETSTATIC, name, "indexes", "[[S");
mv.visitVarInsn(ILOAD, 0);
mv.visitInsn(AALOAD);
mv.visitVarInsn(ASTORE, 1);
mv.visitVarInsn(ALOAD, 1);
mv.visitInsn(ARRAYLENGTH);
mv.visitIntInsn(NEWARRAY, T_CHAR);
mv.visitVarInsn(ASTORE, 2);
// 25 more lines
mv.visitMethodInsn(INVOKESPECIAL, "java/lang/String", "<init>", "([C)V", false);
mv.visitInsn(ARETURN);
mv.visitMaxs(0, 0);
mv.visitEnd();
```

DEBUG INFORMATION


```
L4
 LINENUMBER 32 L4
FRAME CHOP 1
 NEW java/lang/String
 DUP
 ALOAD 2
 INVOKESPECIAL java/lang/String.<init> ([C)V
 ARETURN
L7
 LOCALVARIABLE i I L3 L4 3
 LOCALVARIABLE id I L0 L7 0
 LOCALVARIABLE stringIndexes [S L1 L7 1
 LOCALVARIABLE stringChars [C L2 L7 2
```

GRIP

“SQL-like queries on JVM classes metadata using Kotlin DSL.”

QUERIES


```
val classesQuery =
    grip select classes from classpath where isPublic()

val fieldsQuery =
    grip select fields from classesQuery where
        (isPublic() and not(isFinal()))
```

REFLECTION


```
val mirror = grip.classRegistry.getClassMirror(type)

val fieldsWithParameterizedType = mirror.fields.filter {
  it.signature.type is GenericType.Parameterized
}

val methodsWithProvidesAnnotations = mirror.method.filter {
  it.annotations.contains(getType<Provides>())
}
```

TOOLS

- ASMifier
- javap
- ASM Bytecode Outline
- Kotlin Bytecode

ENOUGH THEORY

LET'S WRITE SOMETHING

PARANOID

A STRING OBFUSCATOR FOR ANDROID

https://github.com/MichaelRocks/paranoid

IDEA

Find classes to `@Obfuscate`

Extract string constants

Replace constants with methods

COMPONENTS

API & runtime

Processor

Build plugin

API

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Obfuscate {
}
```

PROCESSOR

Analysis

Modification

Compilation

ANALYSIS

Initialize Grip

Query annotated classes

INITIALIZE GRIP


```
val grip = GripFactory.create(classpath)
```

QUERY ANNOTATED CLASSES


```
val query = grip
    .select(classes)
    .from(inputPath)
    .where(annotatedWith(getType<Obfuscate>()))

val classes = query.execute().types
```

MODIFICATIONS

Process classes

Hook into a class

Hook into a method

Patch bytecode

PROCESS A CLASS


```
val reader = ClassReader(fileSource.readFile(path))
val writer = ClassWriter(reader, COMPUTE_MAXS or COMPUTE_FRAMES)
val patcher = ParanoidClassVisitor(writer)
reader.accept(patcher, ClassReader.SKIP_FRAMES)
```

HOOK INTO A CLASS


```
class ParanoidClassVisitor(
    delegate: ClassVisitor
) : ClassVisitor(ASM5, delegate) {

  override fun visitMethod(
      access: Int, name: String, desc: String, signature: String?,
      exceptions: Array<out String>?
  ): MethodVisitor {
    val visitor =
        super.visitMethod(access, name, desc, signature, exceptions)
    return ParanoidMethodVisitor(visitor, access, name, desc)
  }
}
```

HOOK INTO A METHOD

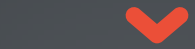

```
class ParanoidMethodVisitor(
    delegate: MethodVisitor, access: Int, name: String, desc: String
) : GeneratorAdapter(ASM5, delegate, access, name, desc) {

  override fun visitLdcInsn(constant: Any) {
    if (constant is String) {
      replaceStringWithMethod(constant)
    } else {
      super.visitLdcInsn(constant)
    }
  }
}
```

PATCH BYTECODE


```
fun replaceStringWithMethod(string: String) {
  val stringId = StringRegistry.registerString(string)
  push(stringId)
  invokeStatic(DEOBFUSCATOR_TYPE, DEOBFUSCATION_METHOD)
}
```

COMPILATION

Create a compiler



Setup a file manager

Create a compilation task

Run compilation

CREATE A COMPILER


```
val compiler = ToolProvider.getSystemJavaCompiler()
val diagnostics = DiagnosticCollector<JavaFileObject>()
```

SETUP A FILE MANAGER


```
val fileManager =
    compiler.getStandardFileManager(diagnostics, null, null)
fileManager.setLocation(SOURCE_PATH, listOf(sourcePath))
fileManager.setLocation(CLASS_OUTPUT, listOf(outputPath))
fileManager.setLocation(CLASS_PATH, classpath)
fileManager.setLocation(PLATFORM_CLASS_PATH, bootClasspath)
```

CREATE A COMPILATION TASK

```
val options = listOf("-g", "-source", "6", "-target", "6")
val units = fileManager.getJavaFileObjects(sourceFile)
val task = compiler.getTask(
    null, fileManager, diagnostics, options, null, units)
```

RUN COMPILATION


```
try {
  if (!task.call()) {
    reportError(diagnostics)
  }
} catch (exception: Exception) {
  reportError(exception)
}
```

BUILD PLUGIN

TRANSFORM API

TRANSFORM SUBCLASS


```
public class ParanoidTransform extends Transform {
  /* Transform magic goes here... */
}
```

TRANSFORMATION NAME


```
@Override
String getName() {
  return "paranoid"
}
```

TRANSFORMATION INPUT


```
@Override
Set<QualifiedContent.ContentType> getInputTypes() {
  return EnumSet.of(QualifiedContent.DefaultContentType.CLASSES)
}
```

TRANSFORMATION SCOPE


```
@Override
Set<QualifiedContent.Scope> getScopes() {
  return EnumSet.of(QualifiedContent.Scope.PROJECT)
}
```

INCREMENTAL?


```
@Override
boolean isIncremental() {
  return false
}
```

EXECUTE PROCESSOR


```
@Override
void transform(
    Context context,
    Collection<TransformInput> inputs,
    Collection<TransformInput> referencedInputs,
    TransformOutputProvider outputProvider,
    boolean isIncremental
) throws IOException, TransformException, InterruptedException {
  /* Invoke your bytecode transformer. */
}
```

REGISTER TRANSFORMATION


```
project.registerTransform(new ParanoidTransform())
```

EXAMPLE


```
@Obfuscate
public class MainActivity extends Activity {
  @Override
  protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);

    final TextView textView = (TextView) findViewById(R.id.textView);
    textView.setText("It works!");
  }
}
```

BEFORE


```
ALOAD 0
LDC 2131427408
INVOKEVIRTUAL io/michaelrocks/paranoid/MainActivity.findViewById
  (I)Landroid/view/View;
CHECKCAST android/widget/TextView
ASTORE 2
ALOAD 2
LDC "It works!"
INVOKEVIRTUAL android/widget/TextView.setText
  (Ljava/lang/CharSequence;)V
```

AFTER


```
ALOAD 0
LDC 2131427408
INVOKEVIRTUAL io/michaelrocks/paranoid/MainActivity.findViewById
  (I)Landroid/view/View;
CHECKCAST android/widget/TextView
ASTORE 2
ALOAD 2
ICONST_0
INVOKESTATIC io/michaelrocks/paranoid/Obfuscator.getString
  (I)Ljava/lang/String;
INVOKEVIRTUAL android/widget/TextView.setText
  (Ljava/lang/CharSequence;)V
```

PITFALLS

Runtime

Debugging

Diagnostics

RUNTIME

APT – **at compile time**

Bytecode weaving - **after compilation**

DEBUGGING

No sources

No debug info

No nothing!

DIAGNOSTICS


```
:samples:sample-android-kotlin:transformClassesWithDexForDebug
Uncaught translation error: com.android.dx.cf.code.SimException:
local 0003: invalid
1 error; aborting
:samples:sample-android-kotlin:transformClassesWithDexForDebug FAILED
```

AND APT?

- Read-only*
- Java only
- Annotations

THAT'S IT

- github.com/MichaelRocks
- me@michaelrocks.io
- @coders_grace