ASM, JEP 457: Class-File API и другие страшные слова

Сергей Владимиров

- 3^3 лет разработки (с 1995 года)
  - 19 лет коммерческой разработки (с 2004 года)
- Ведущий разработчик в Яндекс
- До этого: NetCracker, Одноклассники (VK), Сбербанк, НСПК, Huawei
- Java; JavaScript; InfoSec

JVM, ASM, JEP 457: Class-File API

- Зачем сейчас используют библиотеку ASM
- Какие есть проблемы
- Как эти проблемы решает JEP 457: Class File API

- Как в runtime создать свой Java-класс (и использовать его)
- Как без загрузки класса в Classloader проверить его свойства
- Как сделать простейший статический анализатор для Java

Зачем нужна библиотека ASM?

- Программа-писатель: создание новых классов
- Программа-читатель: статический анализ существующих

Создание новых классов

- [Без ASM] Создание класса:
  - Создаём MyClass.java
  - Вызываем javax.tools.JavaCompier
  - Загружаем в ClassLoader

Создание новых классов

- [Без ASM] Создание класса:
  - Создаём MyClass.java
  - Вызываем javax.tools.JavaCompier
  - Загружаем в ClassLoader

- Траты времени на:
  - анализ и установка classpath
  - парсинг / лексический / синтаксический / статический анализ
  - ограничения компилятора

Создание новых классов

- [Без ASM] Создание класса:
  - Создаём MyClass.java
  - Вызываем javax.tools.JavaCompier
  - Загружаем в ClassLoader
- Траты времени на:
  - анализ и установка classpath
  - парсинг / лексический / синтаксический / статический анализ
  - ограничения компилятора
- [с ASM] Создание класса:
  - Описываем структуру класса и код методов
  - Преобразуем в байты 1-в-1
  - Загружаем в ClassLoader

Создание новых классов

- [Без ASM] Создание класса:
  - Создаём MyClass.java
  - Вызываем javax.tools.JavaCompier
  - Загружаем в ClassLoader
- Траты времени на:
  - анализ и установка classpath
  - парсинг / лексический / синтаксический / статический анализ
  - ограничения компилятора

- [с ASM] Создание класса:
  - Описываем структуру класса и код методов
  - Преобразуем в байты 1-в-1
  - Загружаем в ClassLoader
- Траты кофе на:
  - понимание структуры класса
  - понимание инструкций JVM

Создание новых классов

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Создание новых классов

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

```
javac HelloWorld.java
```

HelloWorld.class

```
00000000  CA FE BA BE 00 00 00 41 00 22 0A 00 02 00 03 07 00 04 0C 00 05 00 06 01 00 10  .......A."..............
0000001A  6A 61 76 61 2F 6C 61 6E 67 2F 4F 62 6A 65 63 74 01 00 06 3C 69 6E 69 74 3E 01  java/lang/Object...<init>.
00000034  00 03 28 29 56 09 00 08 00 09 07 00 0A 0C 00 0B 00 0C 01 00 10 6A 61 76 61 2F  ..()V..............java/
0000004E  6C 61 6E 67 2F 53 79 73 74 65 6D 01 00 03 6F 75 74 01 00 15 4C 6A 61 76 61 2F  lang/System...out...Ljava/
00000068  69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D 3B 08 00 0E 01 00 0D 48 65 6C 6C 6F  io/PrintStream;......Hello
00000082  2C 20 57 6F 72 6C 64 21 0A 00 10 00 11 07 00 12 0C 00 13 00 14 01 00 13 6A 61  , World!..............ja
0000009C  76 61 2F 69 6F 2F 50 72 69 6E 74 53 74 72 65 61 6D 01 00 07 70 72 69 6E 74 6C  va/io/PrintStream...printl
000000B6  6E 01 00 15 28 4C 6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 29 56 07  n...(Ljava/lang/String;)V.
000000D0  00 16 01 00 0A 48 65 6C 6C 6F 57 6F 72 6C 64 01 00 04 43 6F 64 65 01 00 0F 4C  .....HelloWorld...Code...L
000000EA  69 6E 65 4E 75 6D 62 65 72 54 61 62 6C 65 01 00 12 4C 6F 63 61 6C 56 61 72 69  ineNumberTable...LocalVari
00000104  61 62 6C 65 54 61 62 6C 65 01 00 04 74 68 69 73 01 00 0C 4C 48 65 6C 6C 6F 57  ableTable...this...LHelloW
0000011E  6F 72 6C 64 3B 01 00 04 6D 61 69 6E 01 00 16 28 5B 4C 6A 61 76 61 2F 6C 61 6E  orld;...main...([Ljava/lan
00000138  67 2F 53 74 72 69 6E 67 3B 29 56 01 00 04 61 72 67 73 01 00 13 5B 4C 6A 61 76  g/String;)V...args...[Ljav
00000152  61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67 3B 01 00 0A 53 6F 75 72 63 65 46 69 6C  a/lang/String;...SourceFil
0000016C  65 01 00 0F 48 65 6C 6C 6F 57 6F 72 6C 64 2E 6A 61 76 61 00 21 00 15 00 02 00  e...HelloWorld.java.!.....
00000186  00 00 00 00 02 00 01 00 05 00 06 00 01 00 17 00 00 00 2F 00 01 00 01 00 00 00  ................/.......
000001A0  05 2A B7 00 01 B1 00 00 00 02 00 18 00 00 00 06 00 01 00 00 00 01 00 19 00 00  .*......................
000001BA  00 0C 00 01 00 00 00 05 00 1A 00 1B 00 00 00 09 00 1C 00 1D 00 01 00 17 00 00  ........................
000001D4  00 37 00 02 00 01 00 00 00 09 B2 00 07 12 0D B6 00 0F B1 00 00 00 02 00 18 00  .7......................
000001EE  00 00 0A 00 02 00 00 00 03 00 08 00 04 00 19 00 00 00 0C 00 01 00 00 00 09 00  ........................
00000208  1E 00 1F 00 00 00 01 00 20 00 00 00 02 00 21                                    ........ .....!
```

Создание новых классов

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

```
javac HelloWorld.java
javap -v HelloWorld.class
```

Создание новых классов

```
// class version 65.0 (65)
public class HelloWorld {

  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN

  public static main([Ljava/lang/String;)V
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    LDC "Hello, World!"
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
    RETURN
}
```

Создание новых классов

1
```
// class version 65.0 (65)
public class HelloWorld {
```

2
```
  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN

  public static main([Ljava/lang/String;)V
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
```

3
```
    LDC "Hello, World!"
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
    RETURN
}
```

Создание новых классов

Создание новых классов

- Proxy в Spring:
  - `@Async, @Cacheable, @CachePut, @CacheEvict, @Lazy, @Retryable, @Transactional, @Validated`

Создание новых классов

```
@Service
public class FooService {

    @Transactional
    void upsertData(…) {
        /* … */
    }
}
```

Создание новых классов

Создание новых классов

```
@RestController
class BarController {
  @Autowire
  FooService fooService

  void myFunction() {
    fooService.upsertData()
  }
}
```

```
@Service
class FooService {

    @Transactional
    void upsertData(…) {
        /* … */
    }
}
```

Создание новых классов

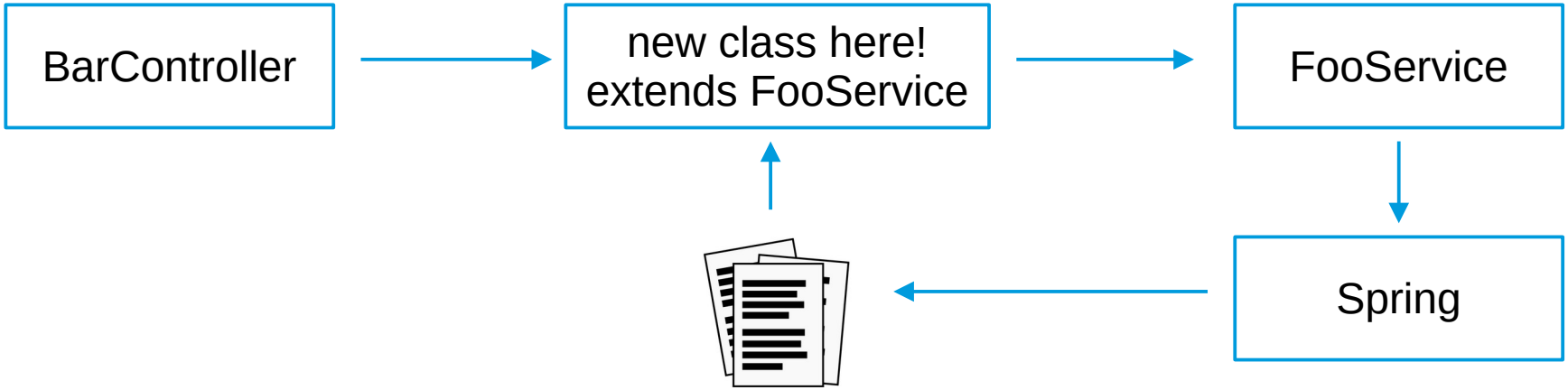
Создание новых классов

Если бы FooService был интерфейсом, то можно было бы использовать `java.lang.reflect.Proxy`

Создание новых классов

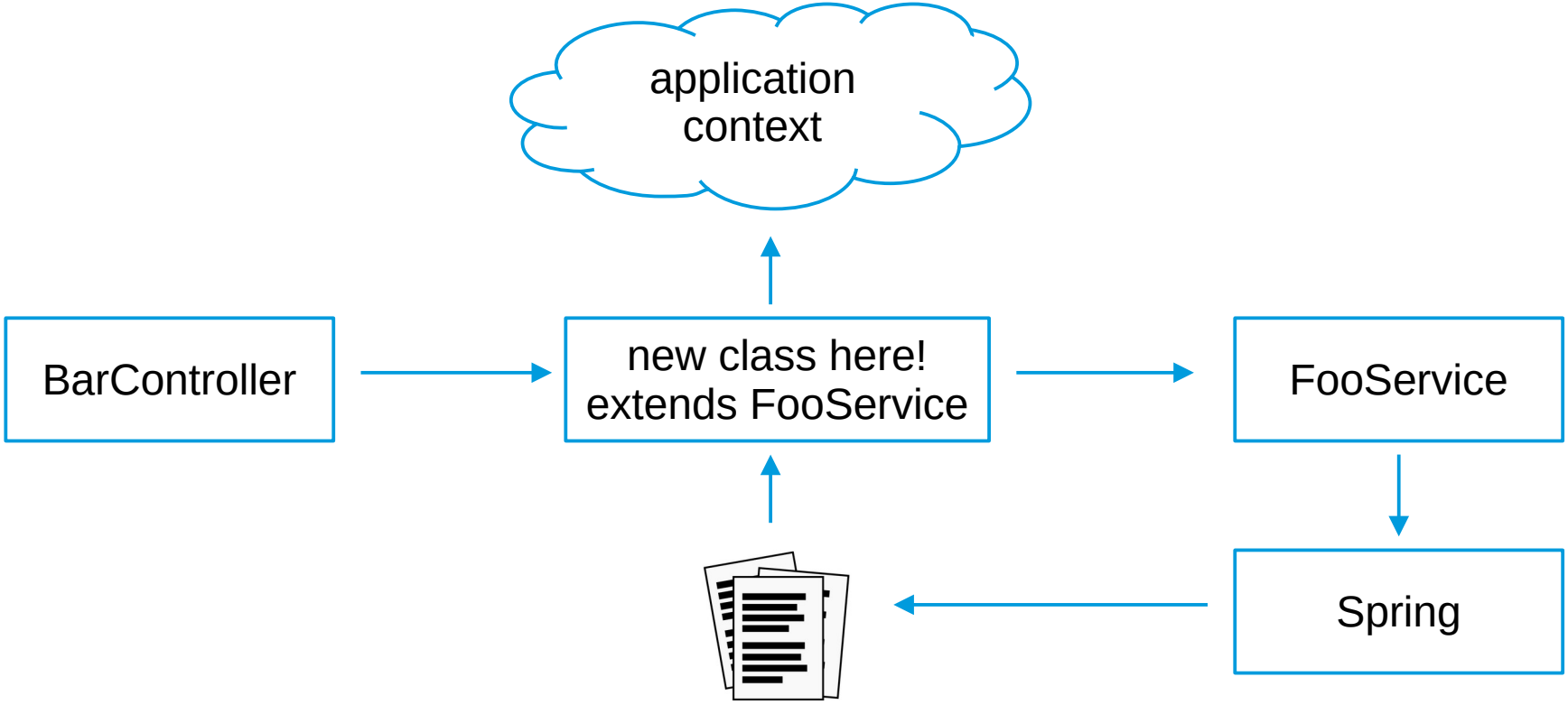
```
public class FooServiceProxy extends FooService {

    private FooService delegate;

    public void setDelegate(FooService delegate) {
        this.delegate = delegate;
    }

    @Override
    void upsertData() {
        // TODO: open TX
        try {
            delegate.upsertData();
        } finally {
            // TODO: close TX
        }
    }
}
```

Создание новых классов



двоичное представление класса

Создание новых классов



двоичное представление класса

Создание новых классов

- Proxy в Spring:
  - `@Async, @Cacheable, @CachePut, @CacheEvict, @Lazy, @Retryable, @Transactional, @Validated`

Создание новых классов

- Proxy в Spring:
  - `@Async, @Cacheable, @CachePut, @CacheEvict, @Lazy, @Retryable, @Transactional, @Validated`
- `JPA / Hibernate:`
  - `lazy entity reference`

Создание новых классов

```
// JPA EntityManager
User user =
  em.getReference(
    User.class, userId)
```

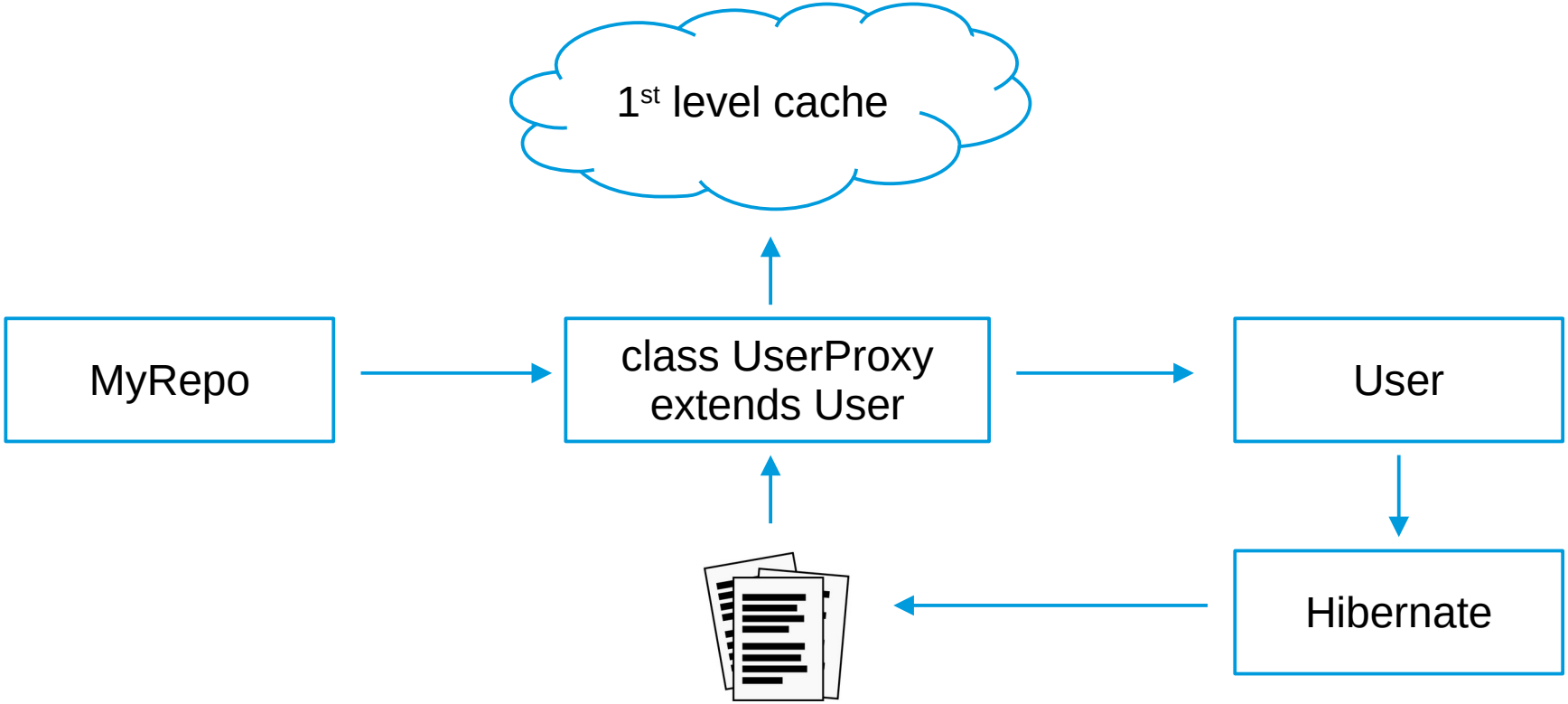
```
class User {
  @Id
  private String userId;
  private String name;
  <…>
}
```

Создание новых классов

Создание новых классов

```
public class UserProxy extends User {
    private final Supplier<User> loader;
    private final String userId;
    private User delegate = null;

    public UserProxy(String userId, Supplier<User> loader) {
        this.userId = userId;
        this.loader = loader;
    }

    @Override
    public String getUserId() {
        return userId;
    }

    @Override
    public String getName() {
        if (delegate == null) this.delegate = loader.get();
        return super.getName();
    }
}
```

Создание новых классов



двоичное представление класса

Создание новых классов

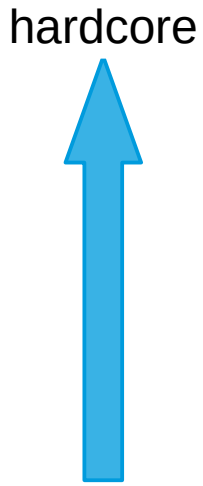
- Proxy в Spring:
  - `@Async, @Cacheable, @CachePut, @CacheEvict, @Lazy, @Retryable, @Transactional, @Validated`
- `JPA / Hibernate:`
  - `lazy entity reference`

Создание новых классов

- Какие инструменты можно для этого использовать, если мы хотим в ***runtime*** получить новый класс?
  - Генерировать двоичный байт-код самому
  - Использовать библиотеку ASM или Class-File API
  - Использовать ByteBuddy, CGLib, Javassist и пр.
  - Генерировать Java-код и вызывать Compiler API

Создание новых классов

- Какие инструменты можно для этого использовать, если мы хотим в ***runtime*** получить новый класс?



  - Генерировать двоичный байт-код самому
  - Использовать библиотеку ASM или Class-File API
  - Использовать ByteBuddy, CGLib, Javassist и пр.
  - Генерировать Java-код и вызывать Compiler API

Создание новых классов

- Сложности:
  - binary class file structure
  - constant pool
  - stack map frames [optimization]

4.1. The `ClassFile` Structure

A `class` file consists of a single `ClassFile` structure:

```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

Создание новых классов

- Сложности:
  - binary class file structure
  - constant pool
  - stack map frames [optimization]
- Библиотека ASM успешно скрывает эти проблемы и от вас, и от других библиотек

4.1. The `ClassFile` Structure

A `class` file consists of a single `ClassFile` structure:

```
ClassFile {
    u4             magic;
    u2             minor_version;
    u2             major_version;
    u2             constant_pool_count;
    cp_info        constant_pool[constant_pool_count-1];
    u2             access_flags;
    u2             this_class;
    u2             super_class;
    u2             interfaces_count;
    u2             interfaces[interfaces_count];
    u2             fields_count;
    field_info     fields[fields_count];
    u2             methods_count;
    method_info    methods[methods_count];
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

Создание новых классов

- Создадим класс, который перехватывает обращение ко всем методам другого класса и что-нибудь делает:
  - печатает имя сервиса в лог
  - открывает и закрывает транзакцию
  - отслеживает время исполнения метода
  - проверяет права пользователя на вызов метода

Создание новых классов

- Создадим класс, который перехватывает обращение ко всем методам другого класса и что-нибудь делает:
  - печатает имя сервиса в лог
  - ~~открывает и закрывает транзакцию~~
  - ~~отслеживает время исполнения метода~~
  - ~~проверяет права пользователя на вызов метода~~

Создание прокси-класса: CGLib

```
class MyMethodInterceptor implements MethodInterceptor {

    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy proxy)
                                                                        throws Throwable {

        System.out.println("Вызван метод: " + method.getName());

        return proxy.invokeSuper(obj, args);

    }

}

Enhancer enhancer = new Enhancer();

enhancer.setSuperclass(OriginalClass.class);

enhancer.setCallback(new MyMethodInterceptor());

OriginalClass proxyObject = (OriginalClass) enhancer.create();

proxyObject.someMethod();
```

Создание прокси-класса: ByteBuddy

```
class Interceptor implements InvocationHandler {
    private final Object original;

    public Interceptor(Object original) { /* ... */ }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws /* ... */ {
        System.out.println(method.getName());
        return method.invoke(original, args);
    }
}

Original original = new Original();
Class<?> dynamicType = new ByteBuddy()
        .subclass(Original.class)
        .method(ElementMatchers.any())
        .intercept(InvocationHandlerAdapter.of(new Interceptor(original)))
        .make().load(ProxyDemo.class.getClassLoader()).getLoaded();

Original instance = (Original) dynamicType.getDeclaredConstructor().newInstance();
instance.someMethod();
```

Создание прокси-класса

- Spring AOP использует AspectJ / CGLib
- AspectJ, ByteBuddy и CGLib используют библиотеку ASM «под капотом»
- ASM генерирует byte-код

Создание прокси-класса: ASM

1. Написать примерный Java-класс руками в IDE
2. Скомпилировать его и посмотреть на структуру класса и байт-код методов (напр. с помощью `javap`)
3. Автоматизировать создание данного класса

Создание прокси-класса: ASM

```
public class ProxyClass extends OriginalClass {

    private OriginalClass originalObject = null;

    public void setOriginalObject(OriginalClass originalObject) {
        this.originalObject = originalObject;
    }

    @Override
    public void someMethod() {
        System.out.println("Вызван метод: someMethod");
        super.someMethod();
    }
}
```

Создание прокси-класса: ASM

```
// class version 65.0 (65)
public class ProxyClass extends OriginalClass {

  private LOriginalClass; originalObject

  public <init>()V
    ALOAD 0
    INVOKESPECIAL OriginalClass.<init> ()V
    ALOAD 0
    ACONST_NULL
    PUTFIELD ProxyClass.originalObject
                              : LOriginalClass;
    RETURN
    MAXSTACK = 2
    MAXLOCALS = 1

  public someMethod()V
    GETSTATIC java/lang/System.out
                          : Ljava/io/PrintStream;
    LDC "Вызван метод: someMethod"
    INVOKEVIRTUAL java/io/PrintStream.println
(Ljava/lang/String;)V
    ALOAD 0
    GETFIELD ProxyClass.originalObject
                                       : LOriginalClass;
    INVOKEVIRTUAL OriginalClass.someMethod ()V
    RETURN
    MAXSTACK = 2
    MAXLOCALS = 1

  public setOriginalObject(LOriginalClass;)V
    ALOAD 0
    ALOAD 1
    PUTFIELD ProxyClass.originalObject
                                       : LOriginalClass;
    RETURN
    MAXSTACK = 2
    MAXLOCALS = 2
}
```

Создание прокси-класса: ASM

```
ClassWriter cw = new
ClassWriter(ClassWriter.COMPUTE_FRAMES);

String className = "ProxyClass";
String superClassName =
Type.getInternalName(OriginalClass.class);

cw.visit(Opcodes.V1_8, Opcodes.ACC_PUBLIC,
className, null, superClassName, null);

// поле для хранения оригинального объекта
FieldVisitor fv =
cw.visitField(Opcodes.ACC_PRIVATE,
"originalObject",

Type.getDescriptor(OriginalClass.class),
null, null);
fv.visitEnd();
```

```
// Генерируем конструктор класса
MethodVisitor mv =
cw.visitMethod(Opcodes.ACC_PUBLIC,
"<init>", "()V", null, null);
mv.visitCode();
mv.visitVarInsn(Opcodes.ALOAD, 0);
mv.visitMethodInsn(Opcodes.INVOKESPECIAL,
superClassName, "<init>", "()V", false);

mv.visitVarInsn(Opcodes.ALOAD, 0);
mv.visitVarInsn(Opcodes.ACONST_NULL, 1);
mv.visitFieldInsn(Opcodes.PUTFIELD,
className, "originalObject",
Type.getDescriptor(OriginalClass.class));

mv.visitInsn(Opcodes.RETURN);
mv.visitMaxs(2, 1);
mv.visitEnd();
```

Создание прокси-класса: ASM

```
// Генерируем метод-перехватчик
mv = cw.visitMethod(Opcodes.ACC_PUBLIC,
"someMethod", "()V", null, null);
mv.visitCode();
mv.visitFieldInsn(Opcodes.GETSTATIC,
"java/lang/System", "out",
"Ljava/io/PrintStream;");
mv.visitLdcInsn("Вызван метод: someMethod");
mv.visitMethodInsn(Opcodes.INVOKEVIRTUAL,
"java/io/PrintStream", "println",
"(Ljava/lang/String;)V", false);

// Вызываем оригинальный метод
mv.visitVarInsn(Opcodes.ALOAD, 0);
mv.visitFieldInsn(Opcodes.GETFIELD, className,
"originalObject",
Type.getDescriptor(OriginalClass.class));
mv.visitMethodInsn(Opcodes.INVOKEVIRTUAL,
Type.getInternalName(OriginalClass.class),
"someMethod", "()V", false);

mv.visitInsn(Opcodes.RETURN);
mv.visitMaxs(2, 1);
mv.visitEnd();

// Генерируем метод для установки оригинального
объекта
mv = cw.visitMethod(Opcodes.ACC_PUBLIC,
"setOriginalObject", "(" +
Type.getDescriptor(OriginalClass.class) + ")V",
null, null);
mv.visitCode();
mv.visitVarInsn(Opcodes.ALOAD, 0);
mv.visitVarInsn(Opcodes.ALOAD, 1);
mv.visitFieldInsn(Opcodes.PUTFIELD, className,
"originalObject",
Type.getDescriptor(OriginalClass.class));
mv.visitInsn(Opcodes.RETURN);
mv.visitMaxs(2, 2);
mv.visitEnd();

// Завершаем генерацию класса
cw.visitEnd();
```

Создание прокси-класса: ASM

```
// Получаем массив байт-кода и загружаем класс
byte[] classBytes = cw.toByteArray();
MyClassLoader classLoader = new MyClassLoader();
Class<?> proxyClass = classLoader.defineClass(className, classBytes);

// Создаем экземпляр proxy-класса
OriginalClass originalObject = new OriginalClass();
Object proxyObject = proxyClass.getDeclaredConstructor().newInstance();

// Устанавливаем оригинальный объект
Method setOriginalObjectMethod =
                    proxyClass.getMethod("setOriginalObject", OriginalClass.class);
setOriginalObjectMethod.invoke(proxyObject, originalObject);

// Вызываем метод на proxy-объекте
proxyObject.someMethod();
```

Поиск классов с аннотацией

- Надо найти все классы внутри некоторого пакета, которые снабжены аннотацией `@Controller`

Поиск классов с аннотацией

- Надо найти все классы внутри некоторого пакета, которые снабжены аннотацией `@Controller`

- org.reflections.reflections (https://github.com/ronmamo/reflections):

```
new Reflections("my.package")
    .getTypesAnnotatedWith(MyAnnotation.class)
```

Проверка наличия аннотации у класса: ASM

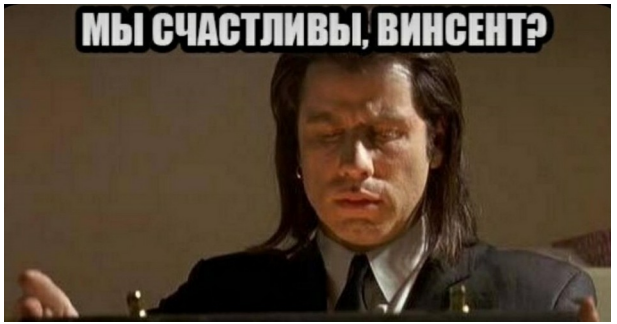
```
new ClassReader( /* byte[] */ ).accept(new ClassVisitor(Opcodes.ASM9) {
    @Override
    public AnnotationVisitor visitAnnotation(String descriptor, boolean visible) {
        if (descriptor.equals("Lorg/springframework/stereotype/Component;")) {
            System.out.println("Found annotation @Component");
        }
        return super.visitAnnotation(descriptor, visible);
    }
}, 0);
```

Использование ASM

- Spring использует AspectJ / CGLib / etc.
- AspectJ, ByteBuddy, CGLib, reflections используют библиотеку ASM «под капотом»
- ASM пишет и читает byte-код

Использование ASM

- Spring использует AspectJ / CGLib / etc.
- AspectJ, ByteBuddy, CGLib, reflections используют библиотеку ASM «под капотом»
- ASM пишет и читает byte-код



Использование ASM

- Spring использует AspectJ / CGLib / etc.
- AspectJ, ByteBuddy, CGLib, reflections используют библиотеку ASM «под капотом»
- ASM пишет и читает byte-код
  - …ломается каждый major-релиз JDK

Использование ASM

```
Caused by: org.springframework.core.NestedIOException: ASM ClassReader
failed to parse class file - probably due to a new Java class file version
that isn't supported yet: URL [jar:...ApplicationConfig.class]; nested
exception is java.lang.IllegalArgumentException: Unsupported class file
major version 65

Caused by: java.lang.IllegalArgumentException: Unsupported class file major
version 65

        at org.springframework.asm.ClassReader.<init>(ClassReader.java:195)
~[spring-core-5.2.2.RELEASE.jar!/:5.2.2.RELEASE]
```

Использование ASM

- Spring с помощью библиотеки ASM ищет кандидатов в бины
  - `@ComponentsScan`
- Для поиска кандидата Spring:
  - Проходит все JAR-файлы в classpath (кроме boostrap)
  - Проходит все class-файлы в них (с учётом имён пакетов)
  - Смотрит на аннотации к классам (`@Component`, `@Service`, etc.)

Использование ASM

- Spring с помощью библиотеки ASM ищет кандидатов в бины
  - `@ComponentsScan`
- Для поиска кандидата Spring:
  - Проходит все JAR-файлы в classpath (кроме boostrap)
  - Проходит все class-файлы в них (с учётом имён пакетов)
  - Смотрит на аннотации к классам (`@Component`, `@Service`, etc.)

  - Spring не загружает каждый класс в Classloader приложения

Использование ASM

- Spring с помощью библиотеки ASM ищет кандидатов в бины
  - `@ComponentsScan`
- Для поиска кандидата Spring:
  - Проходит все JAR-файлы в classpath (кроме boostrap)
  - Проходит все class-файлы в них (с учётом имён пакетов)
  - Смотрит на аннотации к классам (`@Component`, `@Service`, etc.)

  - Spring не загружает каждый класс в Classloader приложения
  - Классы скомпилированные в новых JDK не распарсятся в ASM

Как ломается ASM каждый релиз?

- Если скомпилировали на JDK **18** своё приложение с библиотекой OW2 ASM для JDK **18** (9.2), и запускаете его на JDK **21** — всё хорошо

- Если скомпилировали на JDK **21** своё приложение с библиотекой OW2 ASM для JDK **18** (9.2), и запускаете его на JDK **21+** — всё плохо[1]

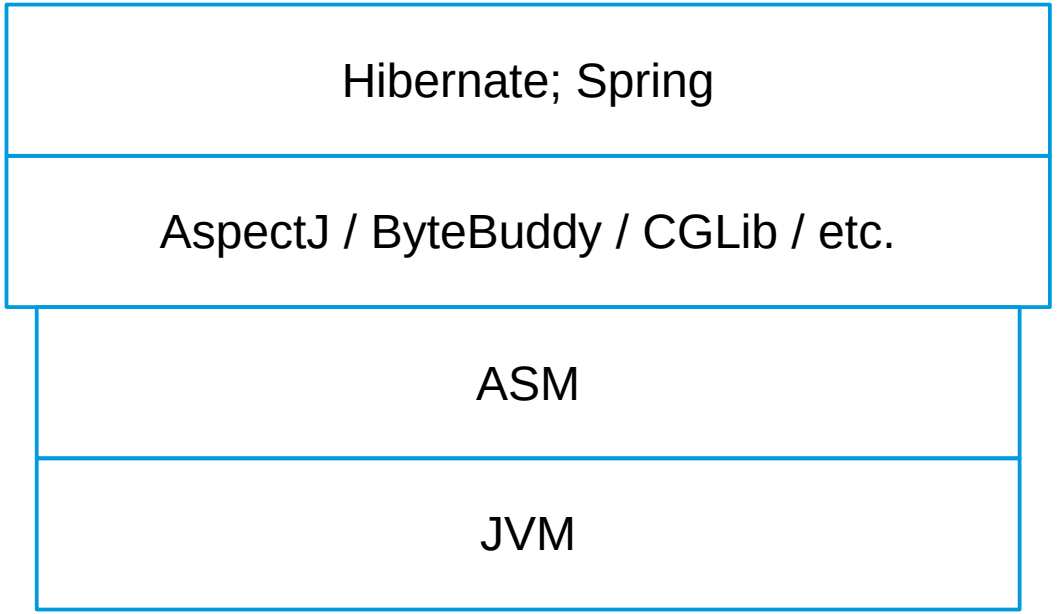
[1] java.lang.IllegalArgumentException: Unsupported class file major version 65
at org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider
.scanCandidateComponents(ClassPathScanningCandidateComponentProvider.java:457)

JEP 457: Class-File API

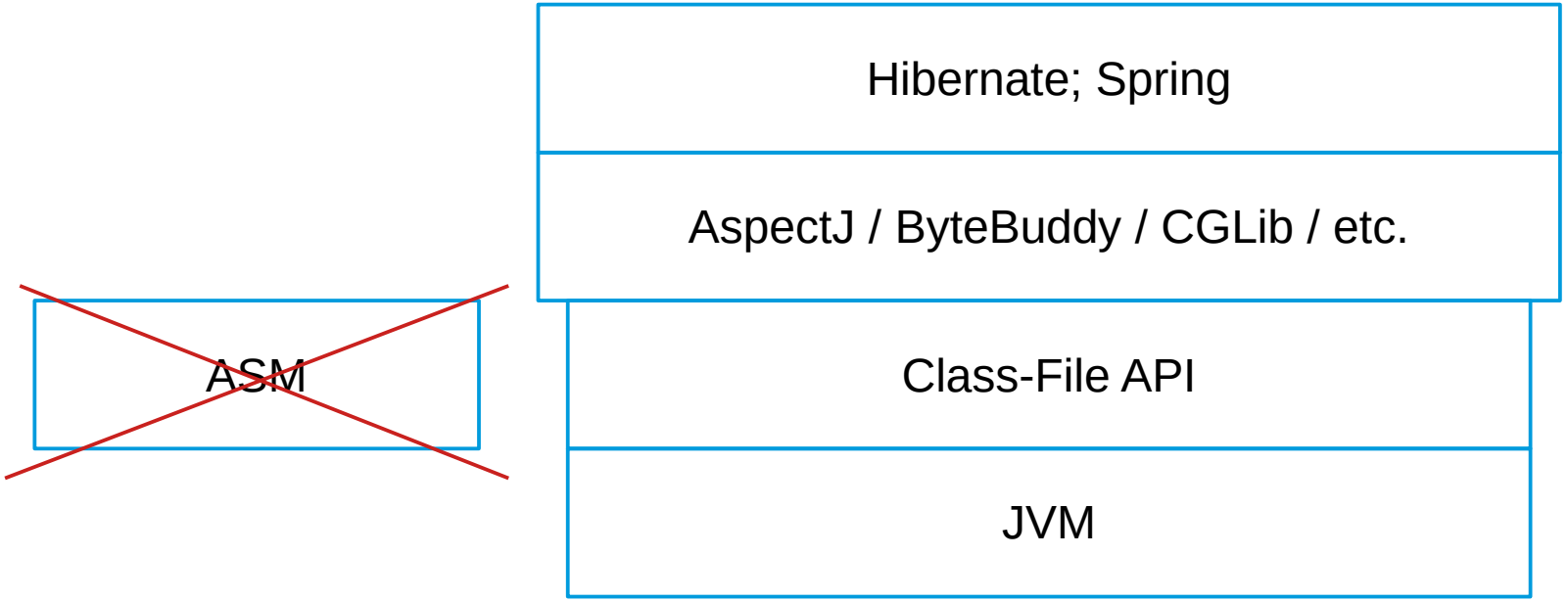
- Проблемы:
  - Библиотека ASM требует обновления каждый major релиз JDK
    - Усложняется тем, что и Spring, и даже OpenJDK содержат shadow-версию OW2 ASM
  - По мнению авторов JEP код и API библиотеки морально устарели

In 2002, the visitor approach used by ASM seemed clever, and was surely more pleasant to use than what came before. However, the Java programming language has improved tremendously since then — with the introduction of lambdas, records, sealed classes, and pattern matching — and the Java Platform now has a standard API for describing class-file constants (java.lang.constant). We can use these features to design an API that is ***more flexible and pleasant to use, less verbose, and less error-prone.***

JEP 457: Class-File API



JEP 457: Class-File API



JEP 457: Class-File API

Release	GA Date	Pr. Support Until
17 (LTS)	September 2021	September 2026
21 (LTS)	September 2023	September 2028
25 (LTS)	September 2025	September 2030

JEP 457: Class-File API

Release	GA Date	Pr. Support Until
17 (LTS)	September 2021	September 2026
21 (LTS)	September 2023	September 2028
25 (LTS)	September 2025	September 2030

JEP 457: Class-File API

Release	GA Date	Pr. Support Until
17 (LTS)	September 2021	September 2026
21 (LTS)	September 2023	September 2028
25 (LTS)	September 2025	September 2030

Пример: Создание класса

- Мы хотим просто создать класс

Создание класса: ASM

```
ClassWriter cw = new ClassWriter(0);
cw.visit(49, ACC_PUBLIC + ACC_SUPER, "Hello", null, "java/lang/Object", null);
cw.visitSource("Hello.java", null);

MethodVisitor mv = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null);
mv.visitVarInsn(ALOAD, 0);
mv.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V", false);
mv.visitInsn(RETURN);
mv.visitMaxs(1, 1);
mv.visitEnd();

cw.visitEnd();
cw.toByteArray();
```

Создание класса: Class File API

```
ClassFile.of().build(ClassDesc.of("Hello"), handler -> {
    handler.withFlags(ClassFile.ACC_PUBLIC);
}); // returns byte[]
```

Создание класса: Class File API

```
var myClassDesc = ClassDesc.of("MethodCallLoggerProxy"); var originalClass = ClassDesc.of("my.package.OriginalClass");

ClassFile.of().build(myClassDesc, handler -> {
    handler.withFlags(ClassFile.ACC_PUBLIC);
    handler.withSuperclass(originalClass);
    handler.withField("originalObject", originalClass, field -> field.withFlags(ClassFile.ACC_PRIVATE));

    // Метод для установки ссылки на оригинальный объект
    handler.withMethodBody("setOriginalObject", ConstantDescs.MTD_void, ClassFile.ACC_PUBLIC, code -> {
        code.aload(0);
        code.aload(1);
        code.putfield(myClassDesc, "originalObject", originalClass);
        code.return_();
    });

    // Метод перехватчик
    handler.withMethodBody("someMethod", ConstantDescs.MTD_void, ClassFile.ACC_PUBLIC, code -> {
        code.getstatic(ClassDesc.of("java.lang.System"), "out", ClassDesc.of("java.io.PrintStream"));
        code.ldc("Вызван метод someMethod");
        code.invokevirtual(ClassDesc.of("java.io.PrintStream"),"println",MethodTypeDesc.of(ConstantDescs.CD_String));

        code.aload(0);
        code.getfield(myClassDesc, "originalObject", originalClass);
        code.invokevirtual(originalClass, "someMethod", ConstantDescs.MTD_void);
    });
});
```

Создание класса: Class File API

- Более компактный API
- Более простой API («`java/lang/Object`» → «`java.lang.Object`»)
- Больше поведений по умолчанию (конструкторы и пр.)

Class-File API: Статический анализ

- Задача по статическому анализу классов («read only»):
- Найти в JAR файле все вызовы «запрещённых» методов:
  - java.time.LocalDate.now()

Class-File API: Статический анализ

```
void visitClassesInJar(File jar, Consumer<InputStream> classBytesConsumer)
        throws IOException {
    try (var zipFile = new ZipFile(jar.getCanonicalFile())) {
        zipFile.stream()
                .filter(entry -> entry.getName().endsWith(".class"))
                .forEach(entry -> {
                    try (var stream = zipFile.getInputStream(entry)) {
                        classBytesConsumer.accept(stream);
                    } catch (IOException e) {
                        // ignore
                    }
                });
    }
}
```

Ищем вызовы методов: ASM

```
visitClassesInJar(jar, clsBytes -> {
  var reader = new ClassReader(clsBytes);
  reader.accept(new MyClassVisitor(consumer), 0);
});
```

Ищем вызовы методов: ASM

```
public class MyClassVisitor extends ClassVisitor {
  private final BiConsumer<MethodKey, MethodKey> consumer;
  private String className;
  public MyClassVisitor(BiConsumer<MethodKey, MethodKey> consumer) {
    super(Opcodes.ASM9);
    this.consumer = consumer;
  }
  @Override
  public void visit(int version, int access, String name,
      String signature, String superName, String[] interfaces) {
    className = name;
    super.visit(version, access, name, signature, superName, interfaces);
  }
```

Ищем вызовы методов: ASM

```
public MethodVisitor visitMethod( int access, String name, String descriptor,
                                  String signature, String[] exceptions ) {
  var callerKey = new MethodKey(className, name, descriptor);
  return new AnalyzerAdapter(Opcodes.ASM9, className, access, name, descriptor, null) {
    @Override
    public void visitMethodInsn(int opcodeAndSource, String owner, String name,
                                String descriptor, boolean isInterface) {
      consumer.accept(callerKey, new MethodKey(owner, name, descriptor));
      super.visitMethodInsn(opcodeAndSource, owner, name, descriptor, isInterface);
    }
  };
}
```

Ищем аннотированные методы

- ASM:

- `bytes → ClassReader`
- `ClassReader ← ClassVisitor`
- `ClassVisitor ← ~~MethodVisitor~~ AnalyzerAdapter`
- `AnalyzerAdapter.visitMethodInsn()`

- `Class-File API:`

Ищем вызовы методов: Class-File API

```
ClassModel classModel = ClassFile.of().parse(clsBytes.readAllBytes());
for (MethodModel method : classModel.methods()) {
  var callerKey = new MethodKey(classModel.thisClass().asInternalName(),
                                method.methodName(), method.methodType());
  method.code().stream()
                .flatMap(CodeModel::elementStream)
                .filter(InvokeInstruction.class::isInstance)
                .map(InvokeInstruction.class::cast)
                .forEach(ii -> {
                    var callee = new MethodKey(ii.owner(), ii.name(), ii.typeSymbol());
                    consumer.accept(callerKey, callee);
                    break;
                });
}
```

Ищем аннотированные методы

- ASM:
  - `bytes → ClassReader`
  - `ClassReader ← ClassVisitor`
  - `ClassVisitor ← ~~MethodVisitor~~ AnalyzerAdapter`
  - `AnalyzerAdapter.visitMethodInsn()`
- `Class-File API:`
  - `bytes → ClassModel`
  - `ClassModel → methods()`
  - `MethodModel → CodeModel`
  - `CodeModel → CodeElement`

Ищем вызовы методов

```
public void analyzeJar(File jarFile) throws IOException {
    this.collectAllMethodCalls(jarFile, (src, dst) -> {
        if (dst.className().equals("java/time/LocalDate")
                && dst.descriptor().contains("now")) {
            out.println(String.format("Found call from %s.%s to %s.%s",
                                      src.className(), src.name(),
                                      dst.className(), dst.name()));
        }
    });
}
```

Class-File API vs ASM

- Обещают поддерживать при выходе новых версий JDK, включая все новые возможности языка
- Более лаконичный (в основном) API для взаимодействия
- Чуть более низкоуровневый, чем ASM
  - но при этом есть большое число функций-помощников

Class-File API vs ASM

- Обещают поддерживать при выходе новых версий JDK, включая все новые возможности языка
- Более лаконичный (в основном) API для взаимодействия
- Чуть более низкоуровневый, чем ASM
  - но при этом есть большое число функций-помощников

- Ждём широкого внедрения (читай – в Spring) в 2028 после окончания поддержки JDK 21
- Но если вы автор библиотеки или используете ASM — задуматься о переходе уже сейчас

Class-File API vs ASM

- https://openjdk.org/jeps/457 – JEP 457
- https://www.youtube.com/watch?v=pcg-E_qyMOI

  « A Classfile API for the JDK» by Brian Goetz, Oracle

Q & A

Сравнение скорости

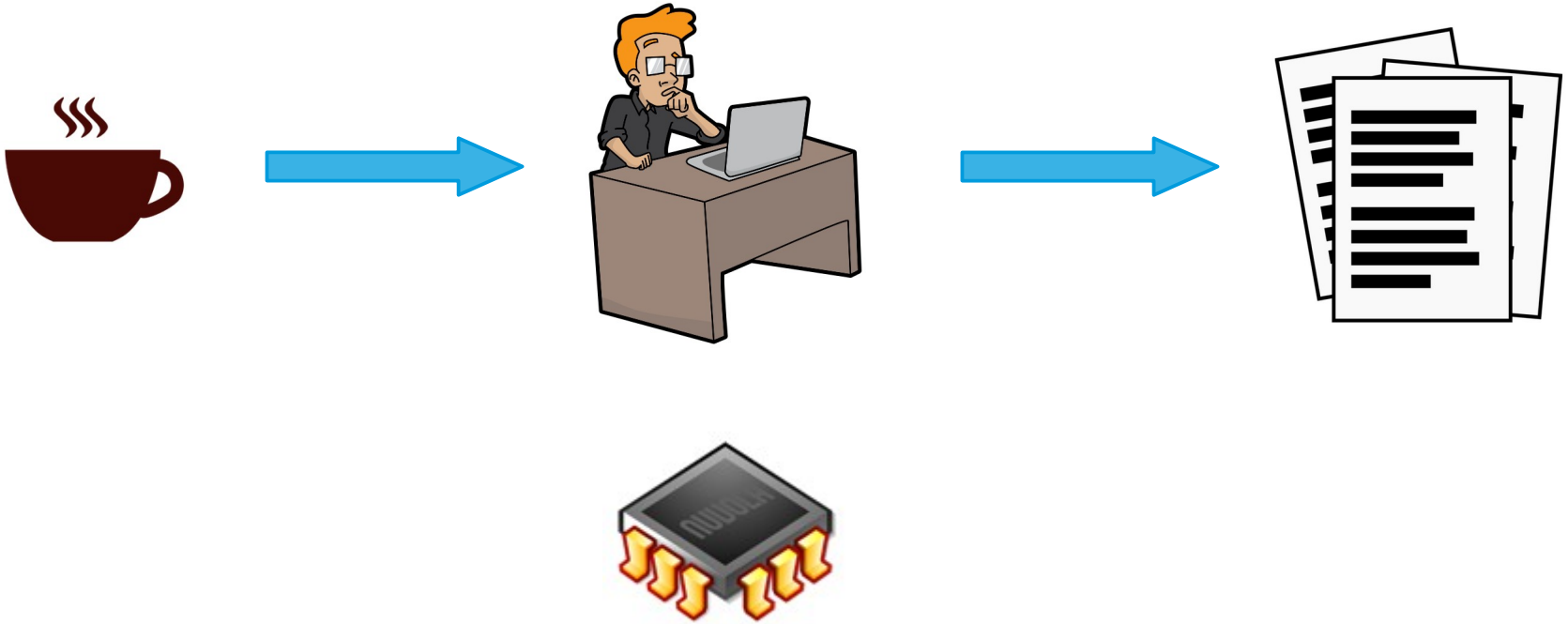
	baseline		Byte Buddy		cglib		Javassist		Java proxy	
trivial class creation	0.003	(0.001)	142.772	(1.390)	515.174	(26.753)	193.733	(4.430)	70.712	(0.645)
interface implementation	0.004	(0.001)	1'126.364	(10.328)	960.527	(11.788)	1'070.766	(59.865)	1'060.766	(12.231)
stub method invocation	0.002	(0.001)	0.002	(0.001)	0.003	(0.001)	0.011	(0.001)	0.008	(0.001)
class extension	0.004	(0.001)	885.983 *5'408.329*	(7.901) *(52.437)*	1'632.730	(52.737)	683.478	(6.735)	–	
super method invocation	0.004	(0.001)	0.004 *0.004*	(0.001) *(0.001)*	0.021	(0.001)	0.025	(0.001)	–	

http://bytebuddy.net/#/tutorial

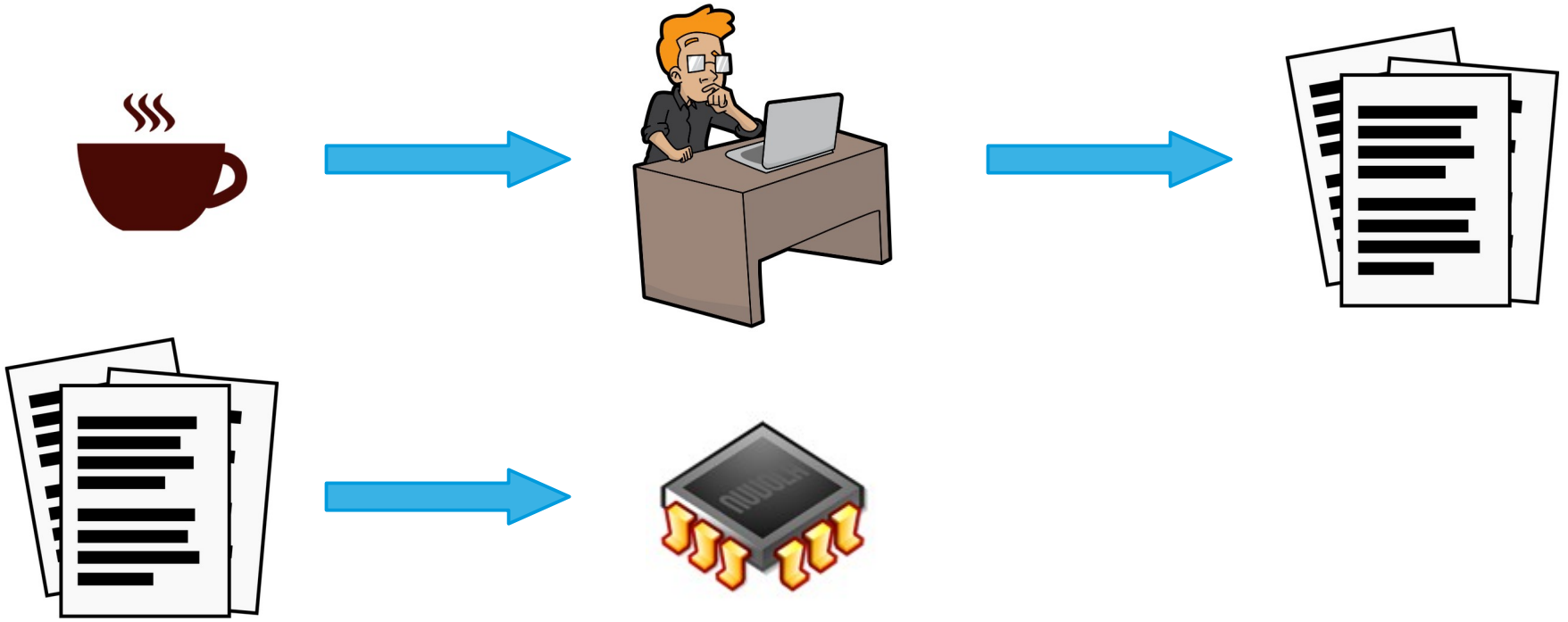
JVM vs «обычный» CPU

JVM vs «обычный» CPU

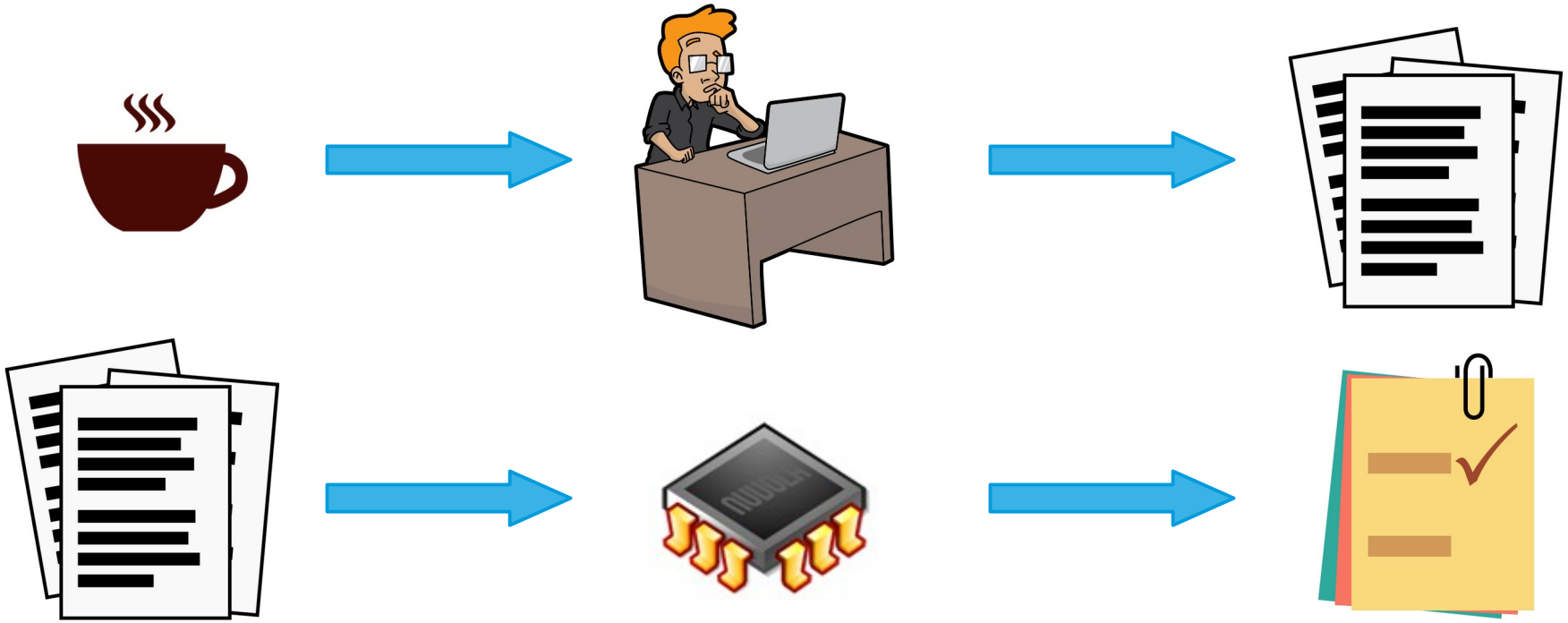
JVM vs «обычный» CPU



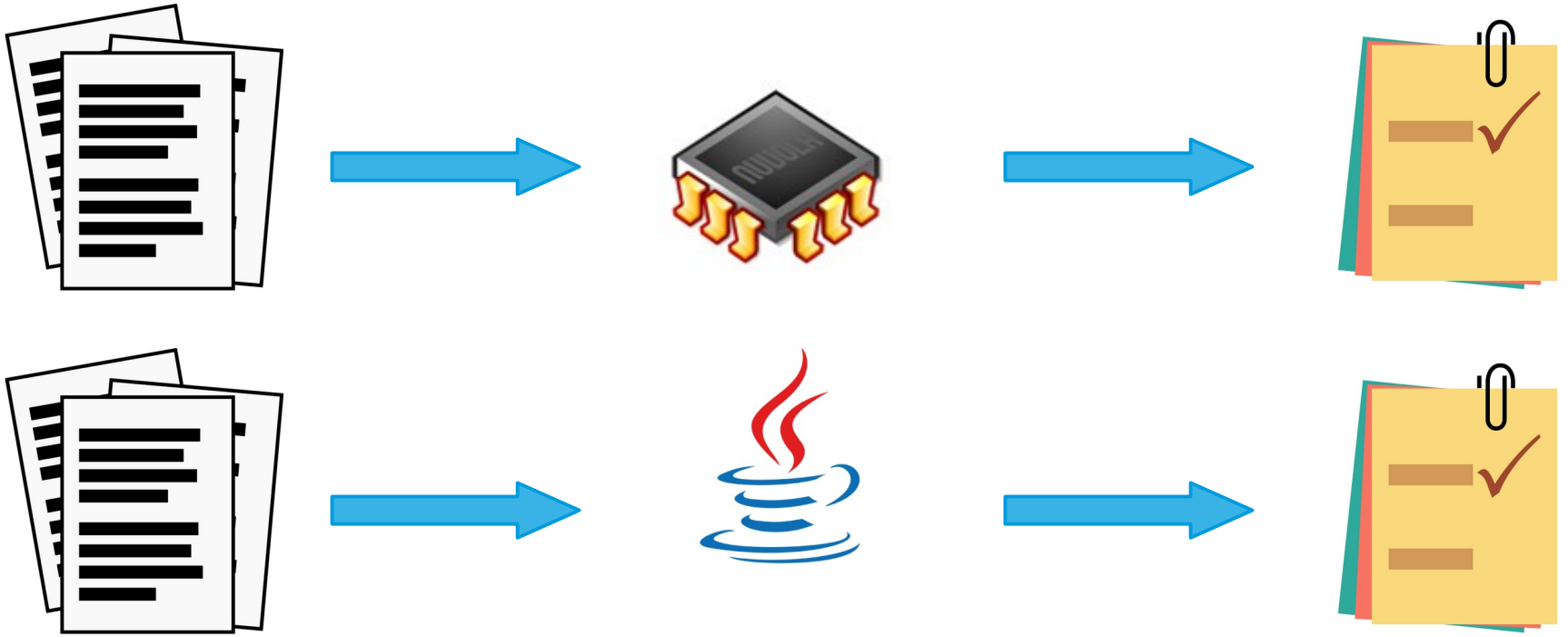
JVM vs «обычный» CPU



JVM vs «обычный» CPU



JVM vs «обычный» CPU



JVM vs «обычный» CPU

JVM vs «обычный» CPU

регистры	стек

JVM vs «обычный» CPU

регистры	стек
нет типов, адресная арифметика, общая память кода и данных	строгая типизация, данные != код, нет адресной арифметики

JVM vs «обычный» CPU

регистры	стек
нет типов, адресная арифметика, общая память кода и данных	строгая типизация, данные != код, нет адресной арифметики
«плоская» модель кода	классы, функции, объекты

JVM vs «обычный» CPU

- «Обычный» CPU:

```
mov     eax, 14
mov     ebx, 10
add     eax, ebx
```

JVM vs «обычный» CPU

- «Обычный» CPU:

```
mov     eax, 14
mov     ebx, 10
add     eax, ebx
```

JVM vs «обычный» CPU

- «Обычный» CPU:

```
mov     eax, 14
mov     ebx, 10
add     eax, ebx
```

- Java-«процессор»:

```
bipush 14
bipush 10
iadd
```

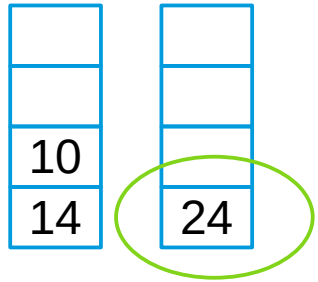
JVM vs «обычный» CPU

- «Обычный» CPU:

```
mov     eax, 14
mov     ebx, 10
add     eax, ebx
```

- Java-«процессор»:

```
bipush 14
bipush 10
iadd
```



JVM vs «обычный» CPU

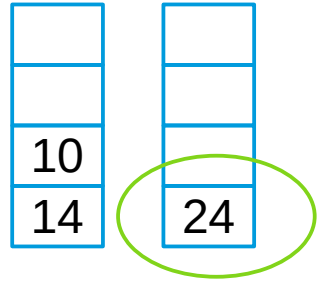
- «Обычный» CPU:

```
mov     eax, 14
mov     ebx, 10
add     eax, ebx
```

- работа с регистрами

- Java-«процессор»:

```
bipush 14
bipush 10
iadd
```



- работа со стеком

JVM vs «обычный» CPU

- «Обычный» CPU:
- сколько разных регистров?
- Java-«процессор»:
- какого размера стек?

JVM vs «обычный» CPU

- «Обычный» CPU:
- ~~сколько разных регистров?~~

- Java-«процессор»:
- ~~какого размера стек?~~

JVM vs «обычный» CPU

- «Обычный» CPU:
- ~~сколько разных регистров?~~
- что можно положить в регистр?

- Java-«процессор»:
- ~~какого размера стек?~~
- что можно положить «на стек»?

JVM vs «обычный» CPU

- «Обычный» CPU:
- ~~сколько разных регистров?~~
- что можно положить в регистр?
- числа, числа, числа (в т. ч. адреса)

- Java-«процессор»:
- ~~какого размера стек?~~
- что можно положить «на стек»?
- числа,… и ссылки на объекты!

JVM vs «обычный» CPU

- «Обычный» CPU:

  числа, числа, числа
  (в т. ч. адреса)

- работа с массивами:
  - `arr[2]`
  - `*(arr + 2)`

- Java-«процессор»:
- есть числа, а есть «ссылки на кучу»

JVM vs «обычный» CPU

- «Обычный» CPU:

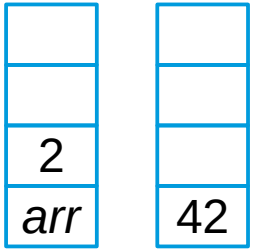
  числа, числа, числа
  (в т. ч. адреса)

- работа с массивами:
  - `arr[2]`
  - `*(arr + 2)`

- Java-«процессор»:
- есть числа, а есть «ссылки на кучу»

(ссылка на массив сверху стека)

```
iconst_2
iaload
```



JVM vs «обычный» CPU

- «Обычный» CPU:

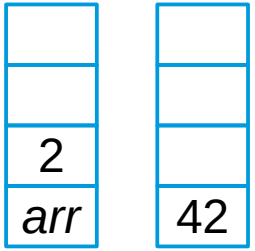
  числа, числа, числа
  (в т. ч. адреса)

- работа с массивами:
  - `arr[2]`
  - `*(arr + 2)`

- Java-«процессор»:
- есть числа, а есть «ссылки на кучу»

(ссылка на массив сверху стека)

```
iconst_2
iaload
```



Нет адресной арифметики – нет Segmentation Fault и атак переполнения буфера!

JVM vs «обычный» CPU

- «Обычный» CPU:

- вызов функций:
  - поместить адрес следующей инструкции на стек
  - передать управление функции используя её адрес (смещение)
  - в конце взять адрес со стека и передать управление обратно

- Java-«процессор»

JVM vs «обычный» CPU

- «Обычный» CPU:
- вызов функций:
  - поместить адрес следующей инструкции на стек
  - передать управление функции используя её адрес (смещение)
  - в конце взять адрес со стека и передать управление обратно

- Java-«процессор»
- вызов методов:
  - поместить ссылку на объект в стек
  - выполнить invokevirtual / invokespecial /
  - в конце вызвать _return

JVM vs «обычный» CPU

- «Обычный» CPU:
- вызов функций:
  - поместить адрес следующей инструкции на стек
  - передать управление функции используя её адрес (смещение)
  - в конце взять адрес со стека и передать управление обратно

- Java-«процессор»
- вызов методов:
  - поместить ссылку на объект в стек
  - выполнить invokevirtual[descriptor]
  - в конце вызвать _return

Код это не данные, а значит нет атаки Java code injection
(впрочем, у нас всё ещё есть SQL, LDAP, XML и куча других интересных мест)

JVM vs «обычный» CPU

- JVM
  - числа это числа, а объекты (ссылки) это объекты (ссылки)
    - нет адресной арифметики
    - стек и все инструкции строго типизированы
  - исполняемый код это не данные
    - загрузить данные как новый* код всё-таки можно, но не «случайно»
  - код находится в методах, методы в объектах, объекты в куче, а описывается всё это классами