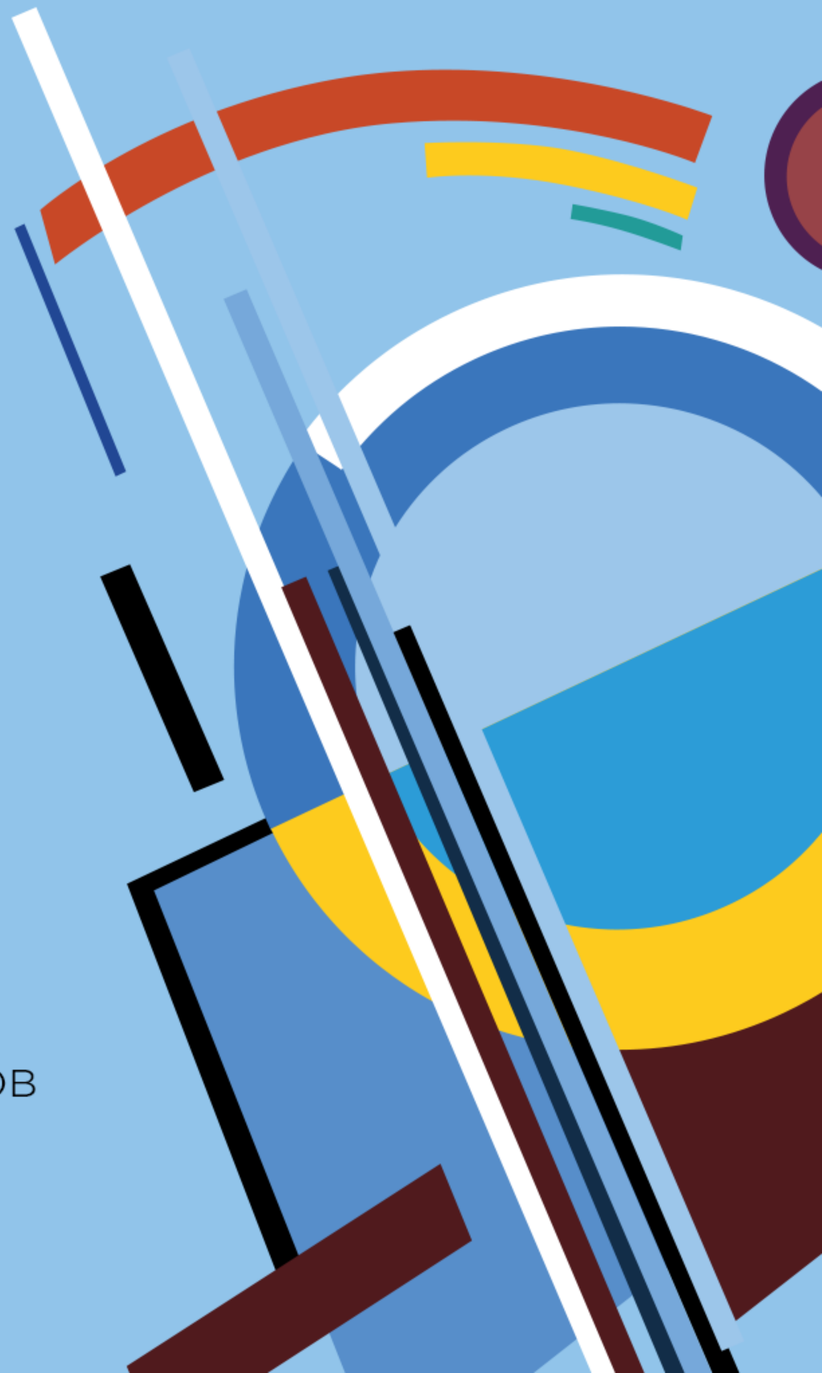
Dynamic proxy и runtime-кодогенерация на Android

Даниил Попов

Apps Conf Moscow 2019

Профессиональная конференция разработчиков мобильных приложений



Коротко обо мне

- В Android-разработке с 2012 года
- Работаю в Mail.ru Group
- Делаю мессенджеры под Android

mail.ru group

Вспомним паттерн

Паттерн Proxy. Проблема

Необходимо контролировать доступ к объекту, не изменяя при этом поведение клиента

Паттерн Proxy. Определение

Заместитель (англ. Proxy) — структурный шаблон проектирования, предоставляющий объект, который контролирует доступ к другому объекту, перехватывая все вызовы.

https://ru.wikipedia.org/wiki/Заместитель_(шаблон_проектирования)

Паттерн Proxy. Виды

- Логирующий прокси
- Удаленный прокси
- “Ленивый” прокси
- Защищающий прокси
- Кэширующий прокси

https://ru.wikipedia.org/wiki/Заместитель_(шаблон_проектирования)

mail.ru group

А можно пример?

Постановка задачи

Нужно логировать вызовы при обращении к репозиторию:

```
public interface UserRepository {
    User getUser(String id);

    void putUser(String id, User user);
}
```

Способы решения

- Голыми руками
- При помощи кодогенерации (например APT)
- Dynamic proxy

Голыми руками

```
public class UserRepositoryProxy implements UserRepository {

    private UserRepository target;

    @Override
    public User getUser(final String id) {
        Log.d(TAG, "getUser " + id);
        return target.getUser(id);
    }

    @Override
    public void putUser(final String id, final User user) {
        Log.d(TAG, "putUser " + id + "; " + user);
        target.putUser(id, user);
    }
}
```

Голыми руками

Плюсы
- Простота реализации
- Не требует сторонних инструментов

Минусы
- Багоопасно
- Много boilerplate-кода
- Рутинная работа

Кодогенерация (APT)

```
@GenerateProxy
public interface UserRepository {
    User getUser(String id);

    void putUser(String id, User user);
}
```

Кодогенерация (APT)

Плюсы

- Boilerplate-код генерируется автоматически
- Менее багоопасно
- Нет рутинной работы

Минусы

- Нужно размечать классы
- Увеличение времени сборки
- Использование сторонних инструментов

Dynamic proxy

```
Proxy.newProxyInstance(UserRepository.class.getClassLoader(),
        new Class[]{UserRepository.class},
        new InvocationHandler() {
            @Override
            public Object invoke(final Object proxy,
                                 final Method method,
                                 final Object[] args) throws Throwable {
                Log.d(TAG, method.getName() + " " + Arrays.toString(args));
                return method.invoke(target, args);
            }
        });
```

Dynamic proxy

Плюсы
- Нет boilerplate-кода
- Менее багоопасно
- Нет рутинной работы
- Не нужно размечать классы
- Не требует сторонних инструментов

Минусы
- ???

mail.ru group

Подробнее о dynamic proxy в Java

Что под капотом?

При вызове `Proxy.newProxyInstance`:

1. Генерируется новый proxy-класс
2. Создается экземпляр этого класса

Свойства proxy-класса

- Класс public, final и не abstract
- Unqualified-имя и пакет не определены
- Наследуется от `java.lang.reflect.Proxy`
- Имплементирует указанные интерфейсы
- Имеет один публичный конструктор с аргументом `InvocationHandler`

Свойства proxy-объекта

- Приводится к указанным интерфейсам
- `instanceof` возвращает `true` для указанных интерфейсов
- Имеет свой `InvocationHandler`
- Все вызовы интерфейсных методов передаются в `InvocationHandler`
- Вызовы `hashCode`, `equals` и `toString` также передаются в `InvocationHandler`

mail.ru group

Кто-то вообще этим пользуется?

Знакомый код?

```
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<Repo>> listRepos(@Path("user") String user);
}

Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .build();

GitHubService service = retrofit.create(GitHubService.class);
```

Знакомый код?

```
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<Repo>> listRepos(@Path("user") String user);
}

Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .build();

GitHubService service = retrofit.create(GitHubService.class);
```

Заглянем внутрь

```
public <T> T create(final Class<T> service) {
    …
    return (T) Proxy.newProxyInstance(service.getClassLoader(),
            new Class<?>[] { service },
            new InvocationHandler() {
                …
                @Override
                public Object invoke(Object proxy,
                                     Method method,
                                     Object[] args) throws Throwable {
                    …
                    return loadServiceMethod(method)
                            .invoke(args != null ? args : emptyArgs);
                }
            });
}
```

mail.ru group

Реальные примеры

Логирование вызовов

```
final Foo target;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
        new Class[]{Foo.class},
        new InvocationHandler() {
            @Override
            public Object invoke(final Object proxy,
                                 final Method method,
                                 final Object[] args) throws Throwable {
                Log.d(TAG, method.getDeclaringClass().getName()
                        + "." + method.getName()
                        + " " + Arrays.toString(args));
                return method.invoke(target, args);
            }
        });
```

Логирование вызовов

```
final Foo target;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
        new Class[]{Foo.class},
        new InvocationHandler() {
            @Override
            public Object invoke(final Object proxy,
                                 final Method method,
                                 final Object[] args) throws Throwable {
                Log.d(TAG, method.getDeclaringClass().getName()
                        + "." + method.getName()
                        + " " + Arrays.toString(args));
                return method.invoke(target, args);
            }
        });
```

Логирование вызовов

```
final Foo target;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
        new Class[]{Foo.class},
        new InvocationHandler() {
            @Override
            public Object invoke(final Object proxy,
                                 final Method method,
                                 final Object[] args) throws Throwable {
                Log.d(TAG, method.getDeclaringClass().getName()
                        + "." + method.getName()
                        + " " + Arrays.toString(args));
                return method.invoke(target, args);
            }
        });
```

Замер времени исполнения

```
import static android.os.SystemClock.elapsedRealtimeNanos;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
    new Class[]{Foo.class},
    new InvocationHandler() {
        @Override
        public Object invoke(final Object proxy,
                             final Method method,
                             final Object[] args) throws Throwable {
            long startTime = elapsedRealtimeNanos();
            Object result = method.invoke(target, args);
            long elapsed = (elapsedRealtimeNanos() - startTime) / 1_000_000;
            Log.d(TAG, "Elapsed " + elapsed + " ms");
            return result;
        }
    });
```

Замер времени исполнения

```
import static android.os.SystemClock.elapsedRealtimeNanos;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
    new Class[]{Foo.class},
    new InvocationHandler() {
        @Override
        public Object invoke(final Object proxy,
                             final Method method,
                             final Object[] args) throws Throwable {
            long startTime = elapsedRealtimeNanos();
            Object result = method.invoke(target, args);
            long elapsed = (elapsedRealtimeNanos() - startTime) / 1_000_000;
            Log.d(TAG, "Elapsed " + elapsed + " ms");
            return result;
        }
    });
```

Замер времени исполнения

```
import static android.os.SystemClock.elapsedRealtimeNanos;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
    new Class[]{Foo.class},
    new InvocationHandler() {
        @Override
        public Object invoke(final Object proxy,
                             final Method method,
                             final Object[] args) throws Throwable {
            long startTime = elapsedRealtimeNanos();
            Object result = method.invoke(target, args);
            long elapsed = (elapsedRealtimeNanos() - startTime) / 1_000_000;
            Log.d(TAG, "Elapsed " + elapsed + " ms");
            return result;
        }
    });
```

Замер времени исполнения

```
import static android.os.SystemClock.elapsedRealtimeNanos;

Proxy.newProxyInstance(Foo.class.getClassLoader(),
    new Class[]{Foo.class},
    new InvocationHandler() {
        @Override
        public Object invoke(final Object proxy,
                             final Method method,
                             final Object[] args) throws Throwable {
            long startTime = elapsedRealtimeNanos();
            Object result = method.invoke(target, args);
            long elapsed = (elapsedRealtimeNanos() - startTime) / 1_000_000;
            Log.d(TAG, "Elapsed " + elapsed + " ms");
            return result;
        }
    });
```

Дефолтный listener

```
public static <T> T empty(Class<T> clazz) {
    return (T) Proxy.newProxyInstance(clazz.getClassLoader(),
            new Class[]{clazz},
            new InvocationHandler() {
                @Override
                public Object invoke(final Object proxy,
                                     final Method method,
                                     final Object[] args) {
                    return defaultValue(method.getReturnType());
                }
            });
}
```

Дефолтный listener

```
private FooListener listener = empty(FooListener.class);

if (listener != null) {
    listener.onBar();
}
```

Дефолтный listener

```
private FooListener listener = empty(FooListener.class);

if (listener != null) {
    listener.onBar();
}
```

Перевод вызова в другой поток

```
public static <T> T ui(final Class<T> clazz, final T delegate) {
    return (T) Proxy.newProxyInstance(clazz.getClassLoader(),
            new Class[]{clazz},
            new InvocationHandler() {
                @Override
                public Object invoke(final Object proxy,
                                     final Method method,
                                     final Object[] args) {
                    invokeOnUi(method, delegate, args);
                    return null;
                }
            });
}
```

Перевод вызова в другой поток

```
private static void invokeOnUi(final Method method,
                               final Object object,
                               final Object... args) {
    mainHandler.post(new Runnable() {
        @Override
        public void run() {
            try {
                method.invoke(object, args);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
    });
}
```

Перевод вызова в другой поток

```
someLibrary.doWork(ui(FooListener.class, new FooListener() {
    @Override
    public void onBar() {
    }
}));
```

Работа с статистикой

```
public interface UserStat {

    void onNameChange(@StatParam("new_name") String newName);

    void onAvatarChange(@StatParam("source") AvatarSource source,
                        @StatParam("w") int width,
                        @StatParam("h") int height);

    void onPasswordChange(@StatParam("len") int length);
}
```

Работа с статистикой

```
new InvocationHandler() {
    @Override
    public Object invoke(final Object proxy,
                         final Method method,
                         final Object[] args) {
        String eventName = method.getName();
        Map<String, Object> eventValues = new HashMap<>();
        final Annotation[][] annotations = method.getParameterAnnotations();
        for (int i = 0; i < annotations.length; i++) {
            StatParam statParam = (StatParam) annotations[i][0];
            eventValues.put(statParam.value(), args[i]);
        }
        sendEvent(eventName, eventValues);
        return null;
    }
});
```

Работа с статистикой

```
new InvocationHandler() {
    @Override
    public Object invoke(final Object proxy,
                         final Method method,
                         final Object[] args) {
        String eventName = method.getName();
        Map<String, Object> eventValues = new HashMap<>();
        final Annotation[][] annotations = method.getParameterAnnotations();
        for (int i = 0; i < annotations.length; i++) {
            StatParam statParam = (StatParam) annotations[i][0];
            eventValues.put(statParam.value(), args[i]);
        }
        sendEvent(eventName, eventValues);
        return null;
    }
});
```

Работа с статистикой

```
new InvocationHandler() {
    @Override
    public Object invoke(final Object proxy,
                         final Method method,
                         final Object[] args) {
        String eventName = method.getName();
        Map<String, Object> eventValues = new HashMap<>();
        final Annotation[][] annotations = method.getParameterAnnotations();
        for (int i = 0; i < annotations.length; i++) {
            StatParam statParam = (StatParam) annotations[i][0];
            eventValues.put(statParam.value(), args[i]);
        }
        sendEvent(eventName, eventValues);
        return null;
    }
});
```

Работа с статистикой

```
new InvocationHandler() {
    @Override
    public Object invoke(final Object proxy,
                         final Method method,
                         final Object[] args) {
        String eventName = method.getName();
        Map<String, Object> eventValues = new HashMap<>();
        final Annotation[][] annotations = method.getParameterAnnotations();
        for (int i = 0; i < annotations.length; i++) {
            StatParam statParam = (StatParam) annotations[i][0];
            eventValues.put(statParam.value(), args[i]);
        }
        sendEvent(eventName, eventValues);
        return null;
    }
});
```

Работа с статистикой

```
new InvocationHandler() {
    @Override
    public Object invoke(final Object proxy,
                         final Method method,
                         final Object[] args) {
        String eventName = method.getName();
        Map<String, Object> eventValues = new HashMap<>();
        final Annotation[][] annotations = method.getParameterAnnotations();
        for (int i = 0; i < annotations.length; i++) {
            StatParam statParam = (StatParam) annotations[i][0];
            eventValues.put(statParam.value(), args[i]);
        }
        sendEvent(eventName, eventValues);
        return null;
    }
});
```

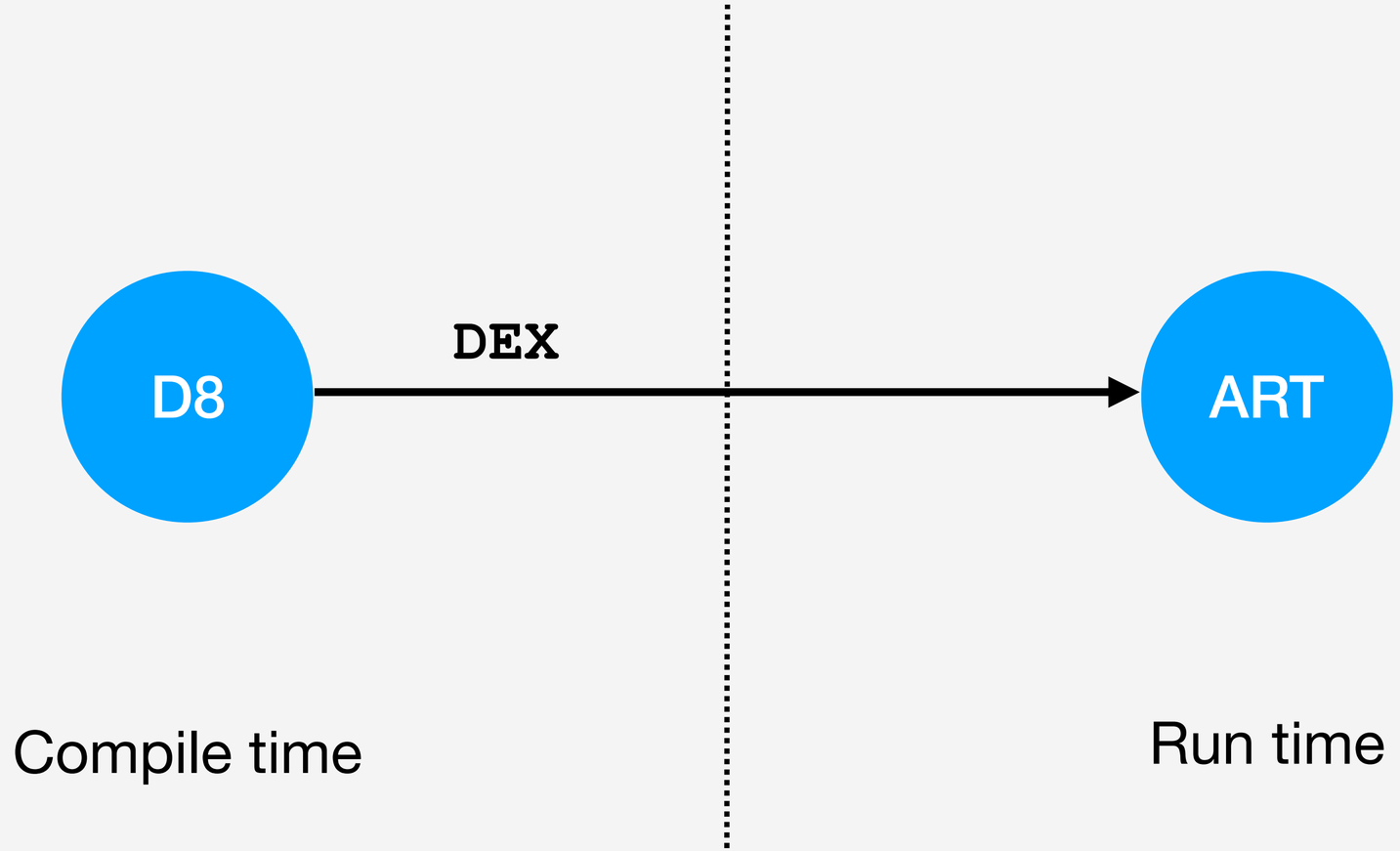
Работа с статистикой

```
stat.onNameChange("John Smith");
stat.onAvatarChange(AvatarSource.CAMERA, 600, 600);
stat.onPasswordChange(64);
```

mail.ru group

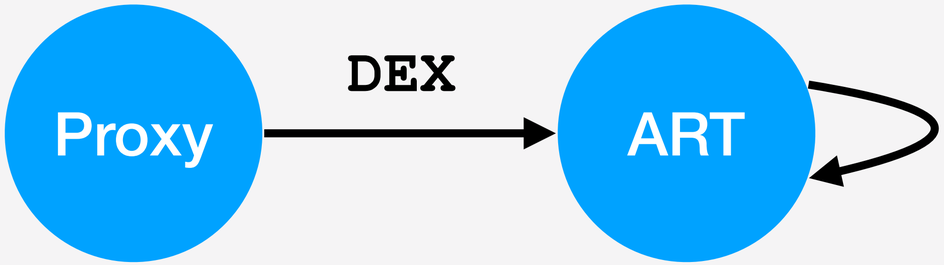
Как proxy устроены внутри

Прямой вызов



Proxy-вызов

Compile time



Run time

Устройство dex-файла

Бинарный файл, который содержит:

- Заголовок
- Таблицу строк
- Описание типов
- Описание прототипов методов
- Описание полей и методов
- Описание классов

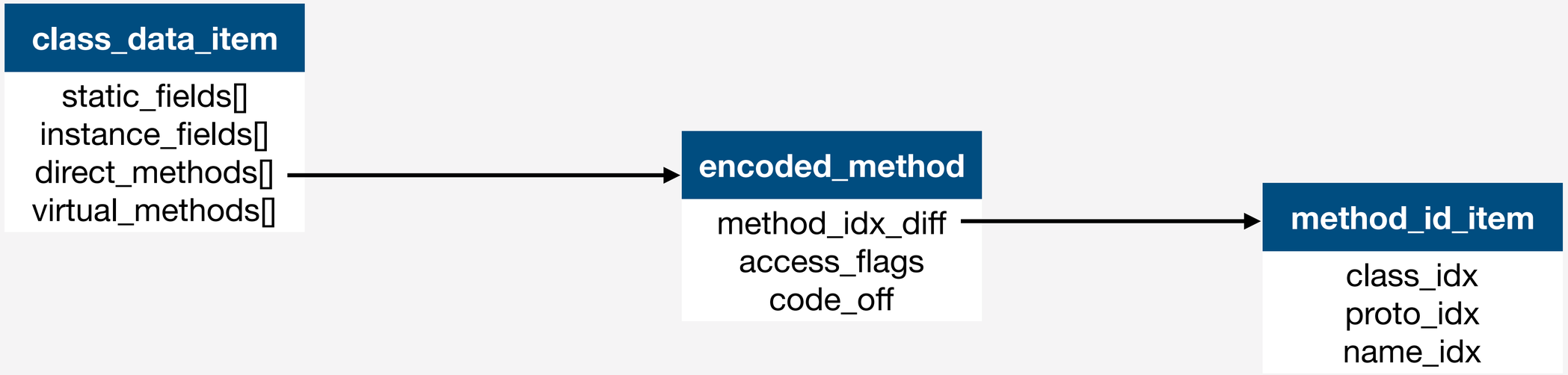
Определение класса

Определение класса

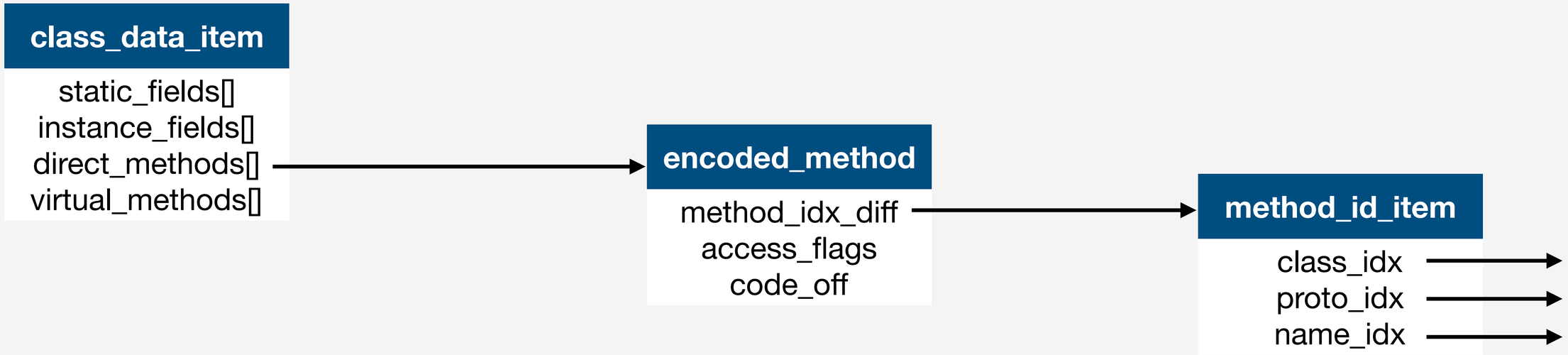
class_data_item
static_fields[]
instance_fields[]
direct_methods[]
virtual_methods[]

Определение класса

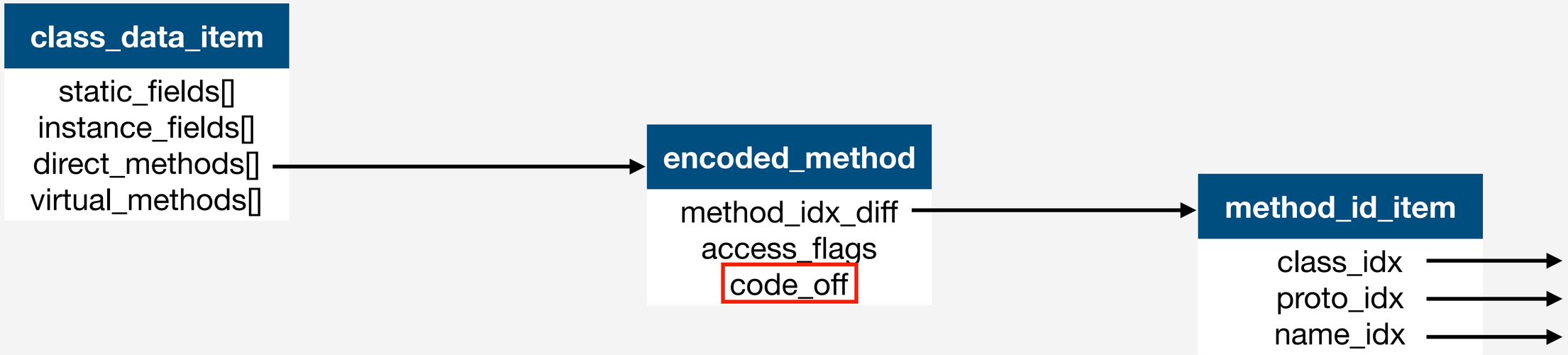
Определение класса



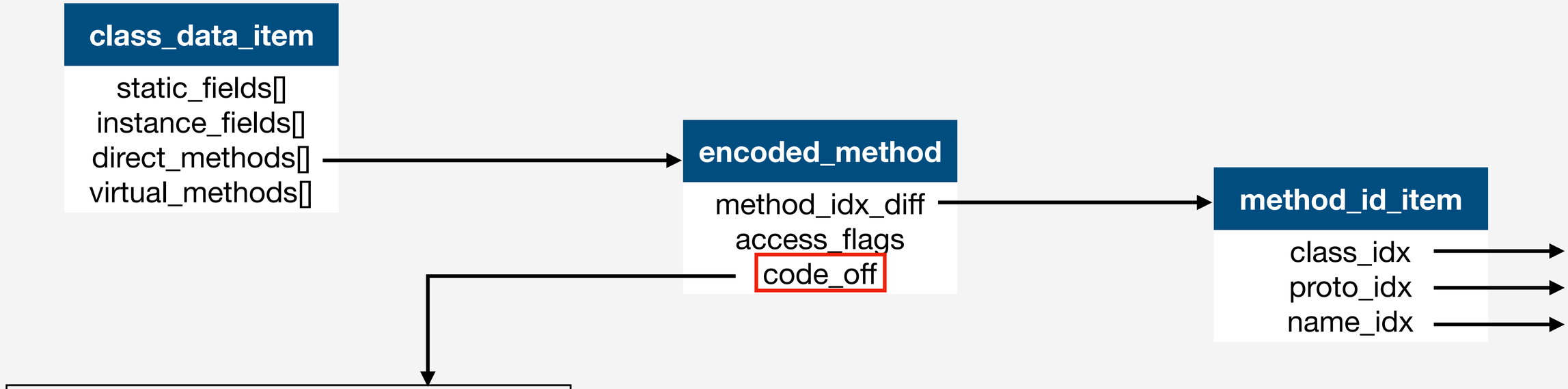
Определение класса



Определение класса



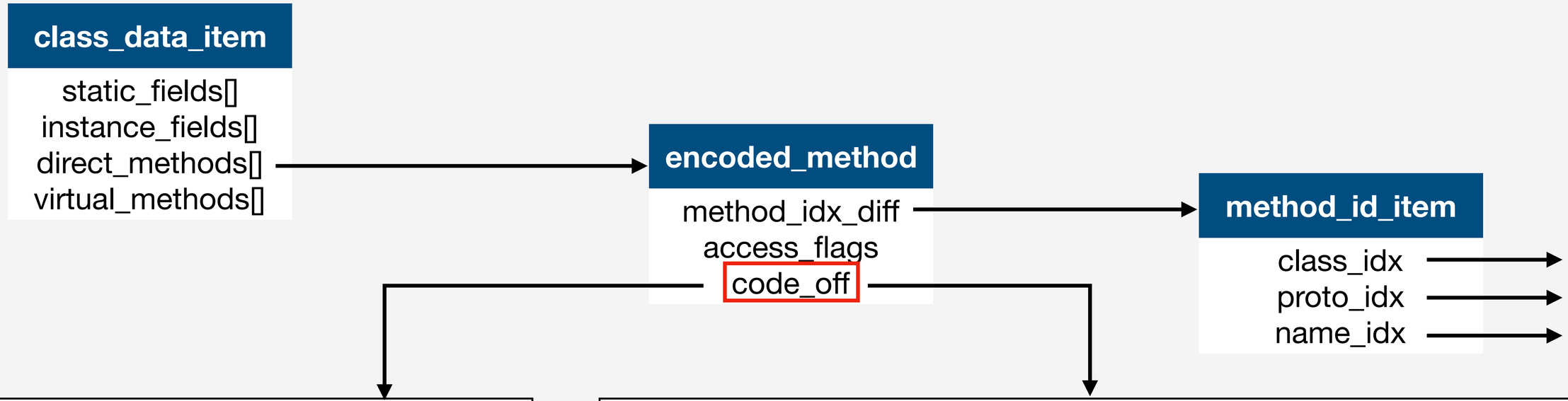
Определение класса




```
ENTRY art_quick_proxy_invoke_handler
    SETUP_SAVE_REFS_AND_ARGS_FRAME_WITH_METHOD_IN_X0
    mov     x2, xSELF
    mov     x3, sp
    bl      artQuickProxyInvokeHandler
    ldr     x2, [xSELF, THREAD_EXCEPTION_OFFSET]
    cbnz    x2, .Lexception_in_proxy
    RESTORE_SAVE_REFS_AND_ARGS_FRAME
    REFRESH_MARKING_REGISTER
    fmov    d0, x0
    ret
.Lexception_in_proxy:
    RESTORE_SAVE_REFS_AND_ARGS_FRAME
    DELIVER_PENDING_EXCEPTION
END art_quick_proxy_invoke_handler
```

arm64

Определение класса




```
ENTRY art_quick_proxy_invoke_handler
    SETUP_SAVE_REFS_AND_ARGS_FRAME_WITH_METHOD_IN_X0
    mov     x2, xSELF
    mov     x3, sp
    bl      artQuickProxyInvokeHandler
    ldr     x2, [xSELF, THREAD_EXCEPTION_OFFSET]
    cbnz    x2, .Lexception_in_proxy
    RESTORE_SAVE_REFS_AND_ARGS_FRAME
    REFRESH_MARKING_REGISTER
    fmov    d0, x0
    ret
.Lexception_in_proxy:
    RESTORE_SAVE_REFS_AND_ARGS_FRAME
    DELIVER_PENDING_EXCEPTION
END art_quick_proxy_invoke_handler
```

arm64

```
DEFINE_FUNCTION art_quick_proxy_invoke_handler
    SETUP_SAVE_REFS_AND_ARGS_FRAME_WITH_METHOD_IN_EAX
    PUSH esp
    pushl %fs:THREAD_SELF_OFFSET
    CFI_ADJUST_CFA_OFFSET(4)
    PUSH ecx
    PUSH eax
    call SYMBOL(artQuickProxyInvokeHandler)
    movd %eax, %xmm0
    movd %edx, %xmm1
    punpckldq %xmm1, %xmm0
    addl LITERAL(16 + FRAME_SIZE_SAVE_REFS_AND_ARGS - FRAME_SIZE_SAVE_REFS_ONLY), %esp
    CFI_ADJUST_CFA_OFFSET(-(16 + FRAME_SIZE_SAVE_REFS_AND_ARGS - FRAME_SIZE_SAVE_REFS_ONLY))
    RESTORE_SAVE_REFS_ONLY_FRAME
    RETURN_OR_DELIVER_PENDING_EXCEPTION
END_FUNCTION art_quick_proxy_invoke_handler
```

x86

Proxy-вызов

- Перебирает стек
- Формирует массив аргументов
- Определяет метод в интерфейсе
- Вызывает handler

```
ENTRY art_quick_proxy_invoke_handler
    SETUP_SAVE_REFS_AND_ARGS_FRAME_WITH_METHOD_IN_X0
    mov     x2, xSELF
    mov     x3, sp
    bl      artQuickProxyInvokeHandler
    ldr     x2, [xSELF, THREAD_EXCEPTION_OFFSET]
    cbnz    x2, .Lexception_in_proxy
    RESTORE_SAVE_REFS_AND_ARGS_FRAME
    REFRESH_MARKING_REGISTER
    fmov    d0, x0
    ret
.Lexception_in_proxy:
    RESTORE_SAVE_REFS_AND_ARGS_FRAME
    DELIVER_PENDING_EXCEPTION
END art_quick_proxy_invoke_handler
```

arm64

mail.ru group

Производительность

Вернемся немного назад

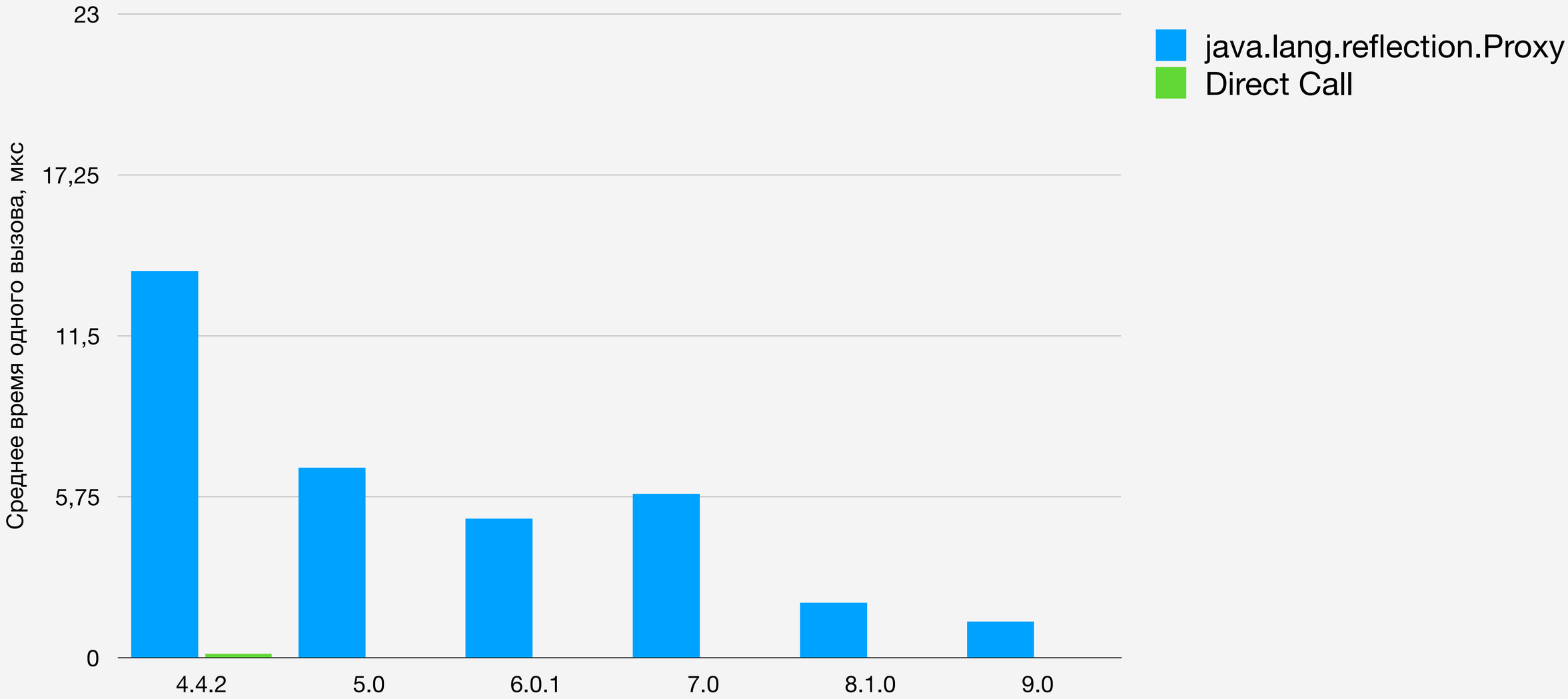
Плюсы
- Нет boilerplate-кода
- Менее багоопасно
- Нет рутинной работы
- Не нужно размечать классы
- Не требует сторонних инструментов

Минусы
- Рефлексия
- Низкая производительность

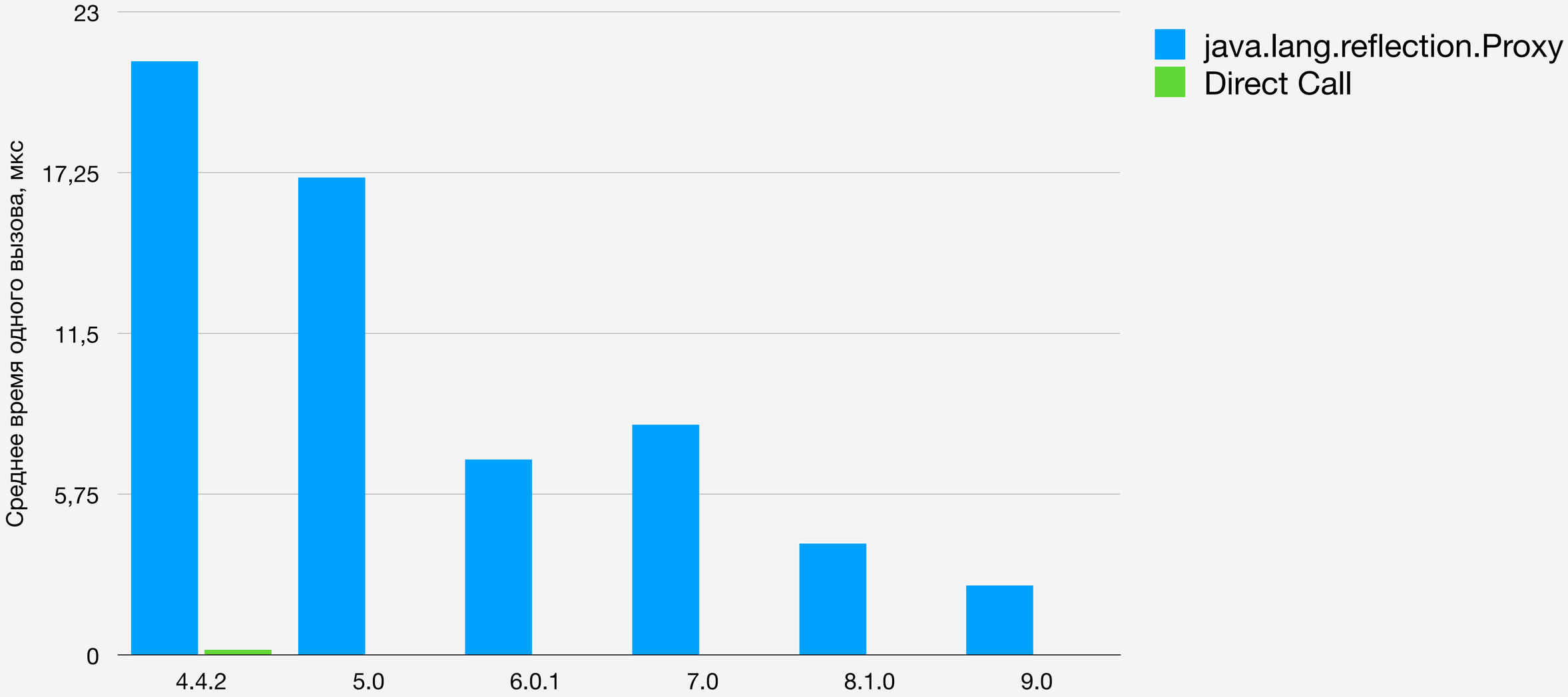
Насколько все плохо?

1. Берем `java.lang.reflect.Proxy` и прямой вызов
2. Выполняем по 1 млн раз
   - Без аргументов
   - 3 примитива
   - 3 ссылочных типа
3. Измеряем среднее время
4. Повторяем на разных мажорных версиях Android

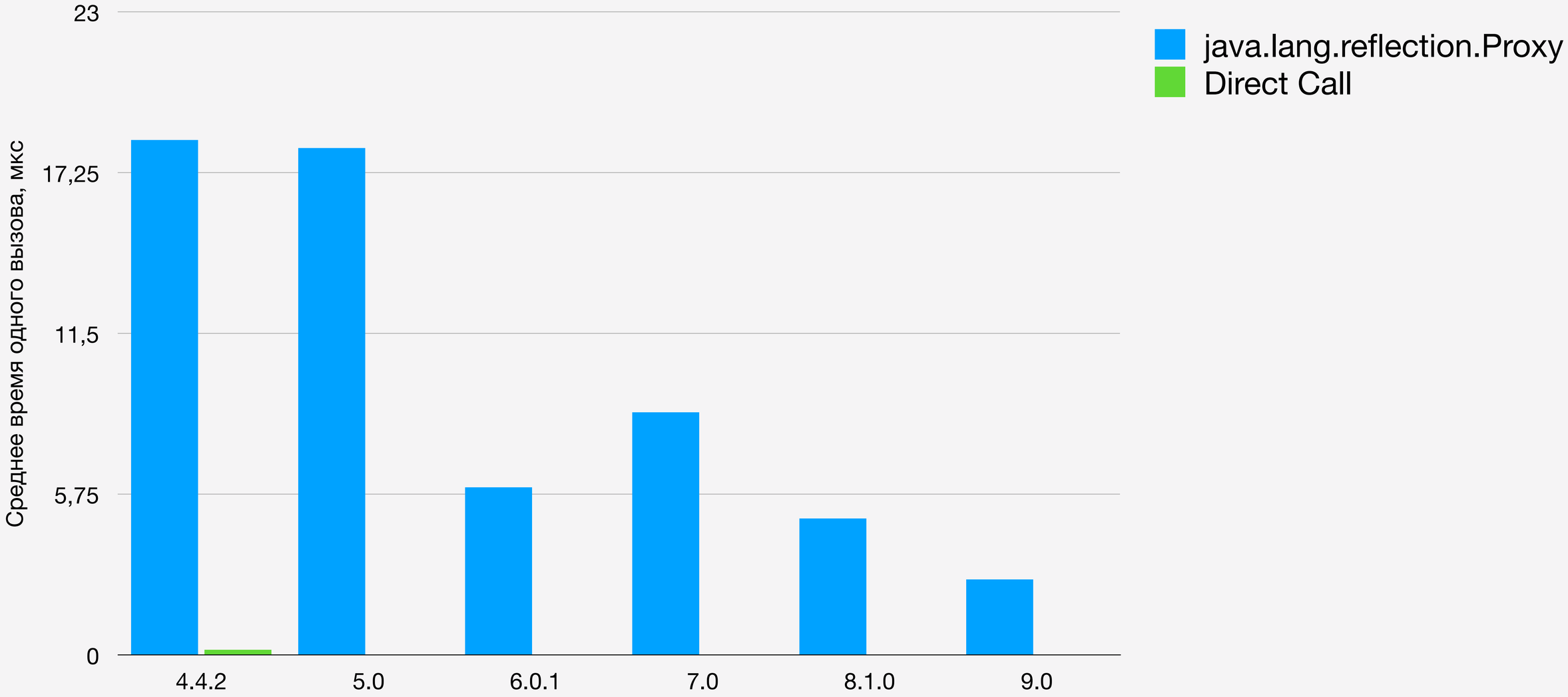
Без аргументов



3 примитива


java.lang.reflection.Proxy
Direct Call
Среднее время одного вызова, мкс
23
17,25
11,5
5,75
0
4.4.2
5.0
6.0.1
7.0
8.1.0
9.0

3 ссылочных типа



mail.ru group

В поисках решения

Что делать?

Что делать?

1. Забить 😢

Что делать?

1. Забить 😢
2. Рефлексия  Кодогенерация

Манипуляции с ClassLoader’ом

defineClass

added in API level 1

```
protected final Class<?> defineClass (String name,
                byte[] b,
                int off,
                int len)
```

Converts an array of bytes into an instance of class `Class`. Before the `Class` can be used it must be resolved.

Oops...

@ mail.ru group

```
/**
 * @since  1.1
 */
protected final Class<?> defineClass(String name,
                                     byte[] b,
                                     int off,
                                     int len) throws ClassFormatError {
    throw new UnsupportedOperationException("can't load this type of class file");
}
```

DexClassLoader

mail.ru group

DexClassLoader

added in API level 3

```
public DexClassLoader (String dexPath,
                String optimizedDirectory,
                String librarySearchPath,
                ClassLoader parent)
```

Creates a `DexClassLoader` that finds interpreted and native code. Interpreted classes are found in a set of DEX files contained in Jar or APK files.

Что мы можем сделать?

1. Сгенерировать byte code для Dalvik
2. Записать его в файл
3. Скормить полученный файл в DexClassLoader
4. Загрузить класс
5. Создать экземпляр класса

Как именно?

```
Foo javaProxy = (Foo) Proxy.newProxyInstance(getClassLoader(),
        new Class[]{Foo.class},
        invocationHandler);

Foo fastProxy = ProxyBuilder.createProxy(getApplicationContext(),
        Foo.class,
        invocationHandler);
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

Обрабатываем методы

```
private void generateMethods() {
    methods = interfaceClass.getMethods();

    for (int i = 0; i < methods.length; i++) {
        final Method method = methods[i];
        generateMethodCode(method, i);
    }
}
```

Обрабатываем методы

```
private void generateMethods() {
    methods = interfaceClass.getMethods();

    for (int i = 0; i < methods.length; i++) {
        final Method method = methods[i];
        generateMethodCode(method, i);
    }
}
```

Обрабатываем методы

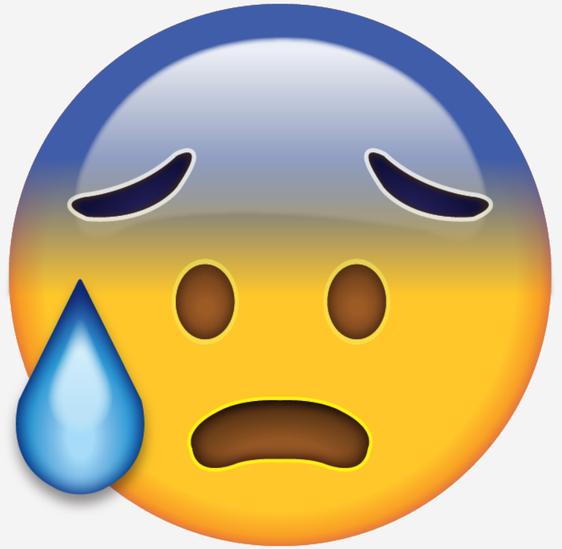
```
private void generateMethods() {
    methods = interfaceClass.getMethods();

    for (int i = 0; i < methods.length; i++) {
        final Method method = methods[i];
        generateMethodCode(method, i);
    }
}
```

generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```

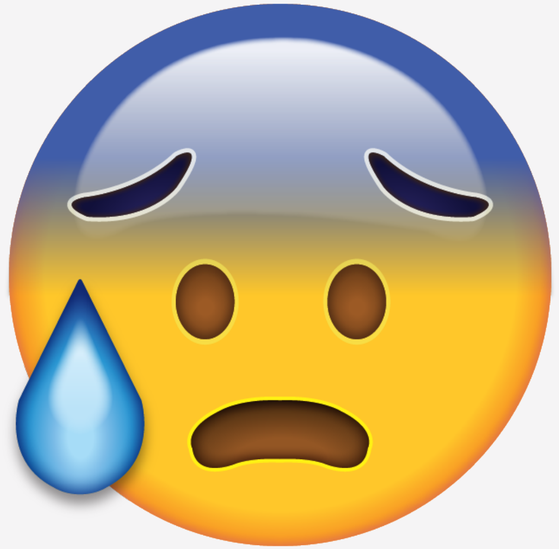
generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```



generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```



generateMethodCode()

```
final DexCode.Builder builder = DexCode.newBuilder()
        .registersSize(registersSize)
        .insSize(insSize)
        .outsSize(4)
        .instruction(igetObject(rHandler, rThis, handlerField))
        .instruction(igetObject(rMethodArray, rThis, methodsField))
        .instruction(const4(rIndex, methodIndex))
        .instruction(agetObject(rMethod, rMethodArray, rIndex))
        .instruction(const4(rArrSize, method.getParameterTypes().length))
        .instruction(newArray(rArgArray, rArrSize,
                dexBuilder.addType(Object[].class)));
```

generateMethodCode()

```
final DexCode.Builder builder = DexCode.newBuilder()
        .registersSize(registersSize)
        .insSize(insSize)
        .outsSize(4)
        .instruction(igetObject(rHandler, rThis, handlerField))
        .instruction(igetObject(rMethodArray, rThis, methodsField))
        .instruction(const4(rIndex, methodIndex))
        .instruction(agetObject(rMethod, rMethodArray, rIndex))
        .instruction(const4(rArrSize, method.getParameterTypes().length))
        .instruction(newArray(rArgArray, rArrSize,
                dexBuilder.addType(Object[].class)));
```

generateMethodCode()

```
final DexCode.Builder builder = DexCode.newBuilder()
        .registersSize(registersSize)
        .insSize(insSize)
        .outsSize(4)
        .instruction(igetObject(rHandler, rThis, handlerField))
        .instruction(igetObject(rMethodArray, rThis, methodsField))
        .instruction(const4(rIndex, methodIndex))
        .instruction(agetObject(rMethod, rMethodArray, rIndex))
        .instruction(const4(rArrSize, method.getParameterTypes().length))
        .instruction(newArray(rArgArray, rArrSize,
                dexBuilder.addType(Object[].class)));
```

generateMethodCode()

```
final DexCode.Builder builder = DexCode.newBuilder()
        .registersSize(registersSize)
        .insSize(insSize)
        .outsSize(4)
        .instruction(igetObject(rHandler, rThis, handlerField))
        .instruction(igetObject(rMethodArray, rThis, methodsField))
        .instruction(const4(rIndex, methodIndex))
        .instruction(agetObject(rMethod, rMethodArray, rIndex))
        .instruction(const4(rArrSize, method.getParameterTypes().length))
        .instruction(newArray(rArgArray, rArrSize,
                dexBuilder.addType(Object[].class)));
```

generateMethodCode()

```
final DexCode.Builder builder = DexCode.newBuilder()
        .registersSize(registersSize)
        .insSize(insSize)
        .outsSize(4)
        .instruction(igetObject(rHandler, rThis, handlerField))
        .instruction(igetObject(rMethodArray, rThis, methodsField))
        .instruction(const4(rIndex, methodIndex))
        .instruction(agetObject(rMethod, rMethodArray, rIndex))
        .instruction(const4(rArrSize, method.getParameterTypes().length))
        .instruction(newArray(rArgArray, rArrSize,
                dexBuilder.addType(Object[].class)));
```

generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```

generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```

generateMethodCode()

mail.ru group

```
for (int i = 0; i < method.getParameterTypes().length; i++) {
    final int rArg = registersSize - argCount + i;
    final Class<?> argType = method.getParameterTypes()[i];
    builder.instruction(const4(rIndex, i));
    if (argType.isPrimitive()) {
        builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                .instruction(moveResultObject(rBoxedArg))
                .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
    } else {
        builder.instruction(aputObject(rArg, rArgArray, rIndex));
    }
}
```

generateMethodCode()

```
for (int i = 0; i < method.getParameterTypes().length; i++) {
    final int rArg = registersSize - argCount + i;
    final Class<?> argType = method.getParameterTypes()[i];
    builder.instruction(const4(rIndex, i));
    if (argType.isPrimitive()) {
        builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                .instruction(moveResultObject(rBoxedArg))
                .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
    } else {
        builder.instruction(aputObject(rArg, rArgArray, rIndex));
    }
}
```

generateMethodCode()

```
for (int i = 0; i < method.getParameterTypes().length; i++) {
    final int rArg = registersSize - argCount + i;
    final Class<?> argType = method.getParameterTypes()[i];
    builder.instruction(const4(rIndex, i));
    if (argType.isPrimitive()) {
        builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                .instruction(moveResultObject(rBoxedArg))
                .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
    } else {
        builder.instruction(aputObject(rArg, rArgArray, rIndex));
    }
}
```

generateMethodCode()

```
for (int i = 0; i < method.getParameterTypes().length; i++) {
    final int rArg = registersSize - argCount + i;
    final Class<?> argType = method.getParameterTypes()[i];
    builder.instruction(const4(rIndex, i));
    if (argType.isPrimitive()) {
        builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                .instruction(moveResultObject(rBoxedArg))
                .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
    } else {
        builder.instruction(aputObject(rArg, rArgArray, rIndex));
    }
}
```

generateMethodCode()

```
for (int i = 0; i < method.getParameterTypes().length; i++) {
    final int rArg = registersSize - argCount + i;
    final Class<?> argType = method.getParameterTypes()[i];
    builder.instruction(const4(rIndex, i));
    if (argType.isPrimitive()) {
        builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                .instruction(moveResultObject(rBoxedArg))
                .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
    } else {
        builder.instruction(aputObject(rArg, rArgArray, rIndex));
    }
}
```

generateMethodCode()

```
for (int i = 0; i < method.getParameterTypes().length; i++) {
    final int rArg = registersSize - argCount + i;
    final Class<?> argType = method.getParameterTypes()[i];
    builder.instruction(const4(rIndex, i));
    if (argType.isPrimitive()) {
        builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
               .instruction(moveResultObject(rBoxedArg))
               .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
    } else {
        builder.instruction(aputObject(rArg, rArgArray, rIndex));
    }
}
```

generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```

generateMethodCode()

```
private void generateMethodCode(final Method method, final int methodIndex) {
    final DexMethod invokeMethod = dexBuilder.addMethod(dexBuilder.addType(InvocationHandler.class),
            dexBuilder.addString("invoke"),
            dexBuilder.addProto(dexBuilder.addType(Object.class), Arrays.asList(
                    dexBuilder.addType(Object.class),
                    dexBuilder.addType(Method.class),
                    dexBuilder.addType(Object[].class)
            )));

    final int argCount = method.getParameterTypes().length;
    final DexMethod dexMethod = dexBuilder.addMethod(proxyType, dexBuilder.addString(method.getName()), getDexProto(method));

    final int rHandler = 0;
    final int rMethod = 1;
    final int rMethodArray = 2;
    final int rIndex = 3;
    final int rArgArray = 4;
    final int rArrSize = 5;
    final int rBoxedArg = 6;
    final int rThis = 7;
    final int registersSize = rThis + argCount + 1;
    final int insSize = argCount + 1;
    final DexCode.Builder builder = DexCode.newBuilder()
            .registersSize(registersSize)
            .insSize(insSize)
            .outsSize(4)
            .instruction(igetObject(rHandler, rThis, handlerField))
            .instruction(igetObject(rMethodArray, rThis, methodsField))
            .instruction(const4(rIndex, methodIndex))
            .instruction(agetObject(rMethod, rMethodArray, rIndex))
            .instruction(const4(rArrSize, method.getParameterTypes().length))
            .instruction(newArray(rArgArray, rArrSize, dexBuilder.addType(Object[].class)));

    for (int i = 0; i < method.getParameterTypes().length; i++) {
        final int rArg = registersSize - argCount + i;
        final Class<?> argType = method.getParameterTypes()[i];
        builder.instruction(const4(rIndex, i));
        if (argType.isPrimitive()) {
            builder.instruction(invokeStatic(rArg, getValueOfMethod(argType)))
                    .instruction(moveResultObject(rBoxedArg))
                    .instruction(aputObject(rBoxedArg, rArgArray, rIndex));
        } else {
            builder.instruction(aputObject(rArg, rArgArray, rIndex));
        }
    }

    builder.instruction(invokeInterface(rHandler, rThis, rMethod, rArgArray, invokeMethod))
            .instruction(returnVoid());

    proxyClassBuilder.virtualMethod(dexMethod, AccessFlags.fromValue(AccessFlags.ACC_PUBLIC), builder.build());
}
```

generateMethodCode()

mail.ru group

```
builder.instruction(invokeInterface(rHandler, rThis, rMethod,
                                    rArgArray, invokeMethod))
       .instruction(returnVoid());
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

Заглянем под капот

```
public static <T> T createProxy(final Context context,
                                final Class<T> clazz,
                                final InvocationHandler handler) {
    final ProxyBuilder<T> builder = new ProxyBuilder<>(clazz, handler);
    builder.prepare();
    builder.generatedFields();
    builder.generateConstructor();
    builder.generateMethods();

    builder.proxyClassBuilder.build();
    builder.writeDex(context);
    return builder.createProxyInstance(context);
}
```

createProxyInstance()

```
private T createProxyInstance(final Context context) {
    final DexClassLoader classLoader = new DexClassLoader(
            getProxyDexFile(context).getAbsolutePath(),
            getCodeCacheDir(context).getAbsolutePath(),
            null,
            ProxyBuilder.class.getClassLoader());
    try {
        Class<?> proxyClass = classLoader.loadClass(getProxyClassName());
        Object proxy = proxyClass
                .getConstructor(InvocationHandler.class, Method[].class)
                .newInstance(invocationHandler, methods);
        return interfaceClass.cast(proxy);
    } catch (ClassNotFoundException | NoSuchMethodException
            | IllegalAccessException | InstantiationException
            | InvocationTargetException e) {
        throw new RuntimeException(e);
    }
}
```

createProxyInstance()

```
private T createProxyInstance(final Context context) {
    final DexClassLoader classLoader = new DexClassLoader(
            getProxyDexFile(context).getAbsolutePath(),
            getCodeCacheDir(context).getAbsolutePath(),
            null,
            ProxyBuilder.class.getClassLoader());
    try {
        Class<?> proxyClass = classLoader.loadClass(getProxyClassName());
        Object proxy = proxyClass
                .getConstructor(InvocationHandler.class, Method[].class)
                .newInstance(invocationHandler, methods);
        return interfaceClass.cast(proxy);
    } catch (ClassNotFoundException | NoSuchMethodException
            | IllegalAccessException | InstantiationException
            | InvocationTargetException e) {
        throw new RuntimeException(e);
    }
}
```

createProxyInstance()

```
private T createProxyInstance(final Context context) {
    final DexClassLoader classLoader = new DexClassLoader(
            getProxyDexFile(context).getAbsolutePath(),
            getCodeCacheDir(context).getAbsolutePath(),
            null,
            ProxyBuilder.class.getClassLoader());
    try {
        Class<?> proxyClass = classLoader.loadClass(getProxyClassName());
        Object proxy = proxyClass
                .getConstructor(InvocationHandler.class, Method[].class)
                .newInstance(invocationHandler, methods);
        return interfaceClass.cast(proxy);
    } catch (ClassNotFoundException | NoSuchMethodException
            | IllegalAccessException | InstantiationException
            | InvocationTargetException e) {
        throw new RuntimeException(e);
    }
}
```

createProxyInstance()

```
private T createProxyInstance(final Context context) {
    final DexClassLoader classLoader = new DexClassLoader(
            getProxyDexFile(context).getAbsolutePath(),
            getCodeCacheDir(context).getAbsolutePath(),
            null,
            ProxyBuilder.class.getClassLoader());
    try {
        Class<?> proxyClass = classLoader.loadClass(getProxyClassName());
        Object proxy = proxyClass
                .getConstructor(InvocationHandler.class, Method[].class)
                .newInstance(invocationHandler, methods);
        return interfaceClass.cast(proxy);
    } catch (ClassNotFoundException | NoSuchMethodException
            | IllegalAccessException | InstantiationException
            | InvocationTargetException e) {
        throw new RuntimeException(e);
    }
}
```

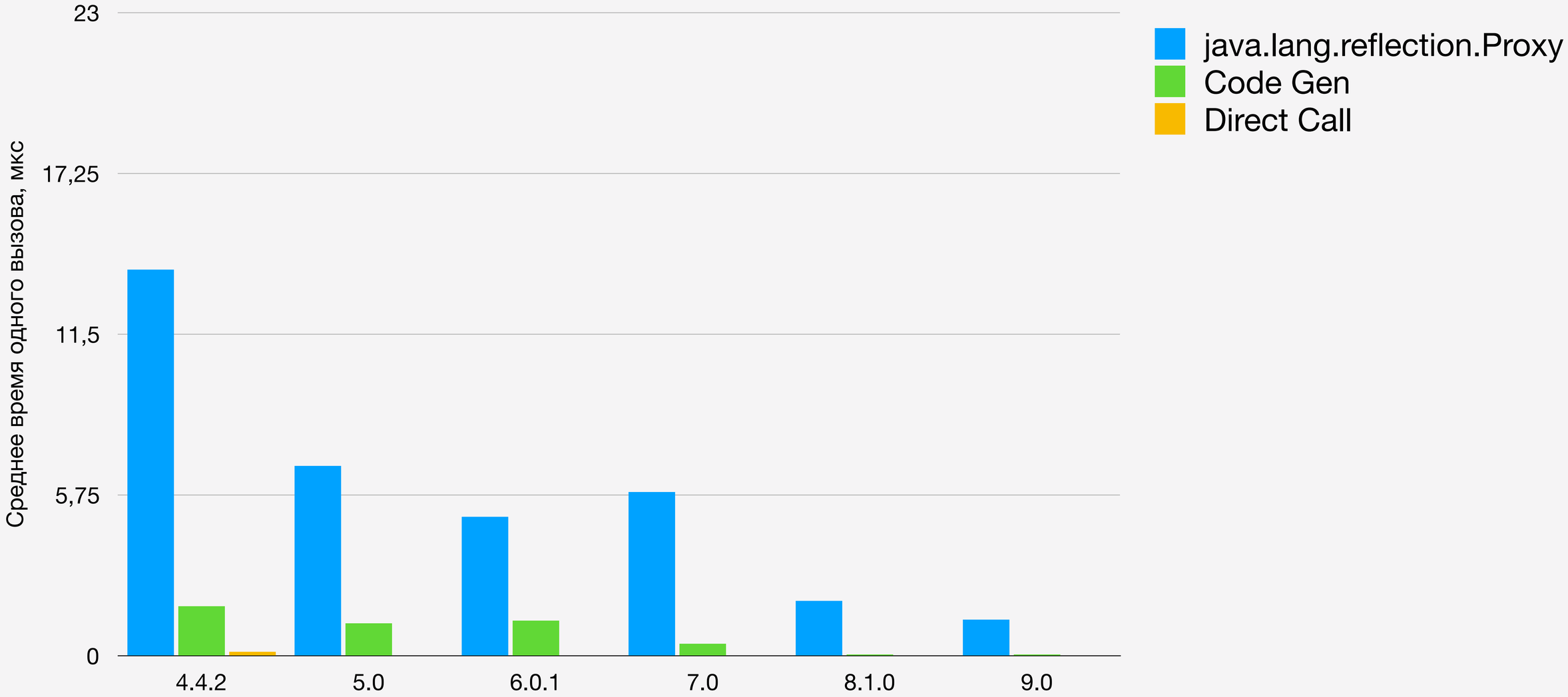
createProxyInstance()

```
private T createProxyInstance(final Context context) {
    final DexClassLoader classLoader = new DexClassLoader(
            getProxyDexFile(context).getAbsolutePath(),
            getCodeCacheDir(context).getAbsolutePath(),
            null,
            ProxyBuilder.class.getClassLoader());
    try {
        Class<?> proxyClass = classLoader.loadClass(getProxyClassName());
        Object proxy = proxyClass
                .getConstructor(InvocationHandler.class, Method[].class)
                .newInstance(invocationHandler, methods);
        return interfaceClass.cast(proxy);
    } catch (ClassNotFoundException | NoSuchMethodException
            | IllegalAccessException | InstantiationException
            | InvocationTargetException e) {
        throw new RuntimeException(e);
    }
}
```

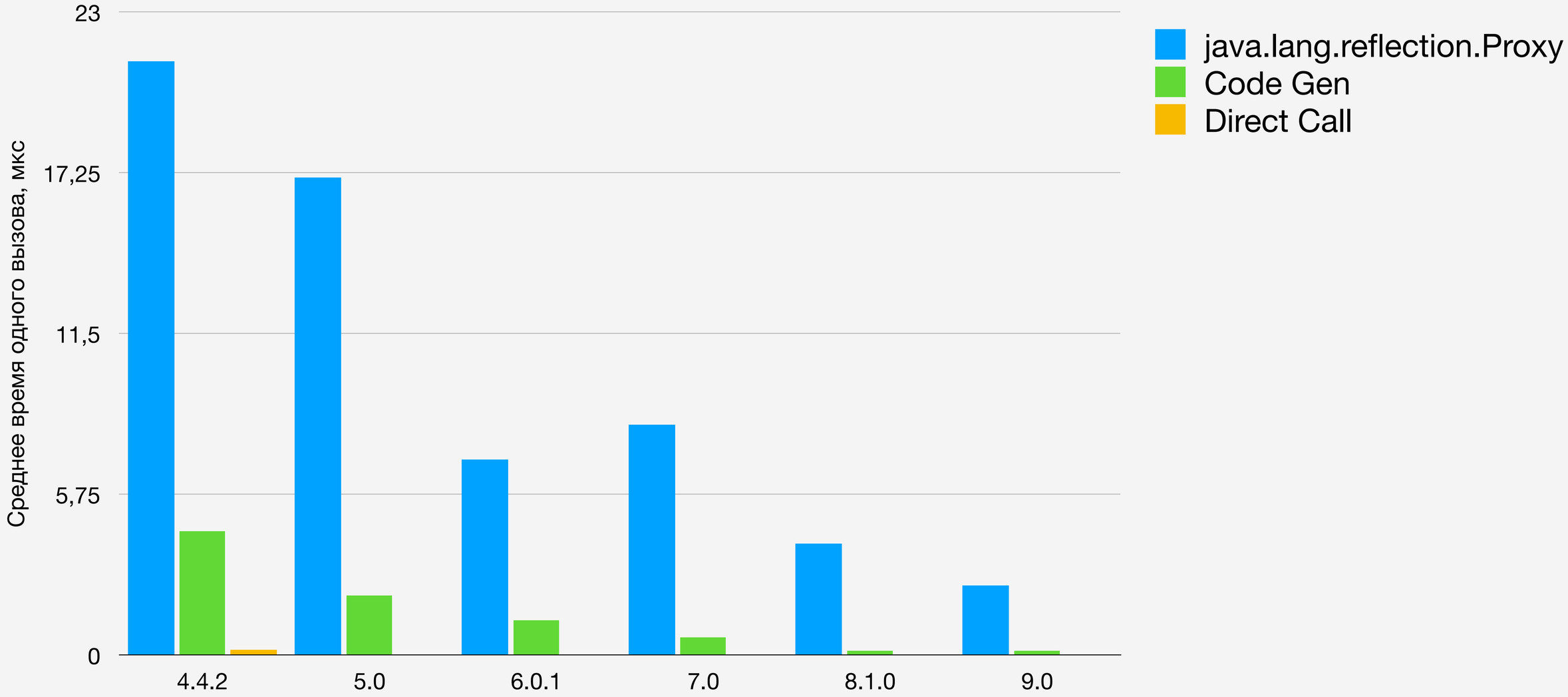
mail.ru group

А что с графиками?

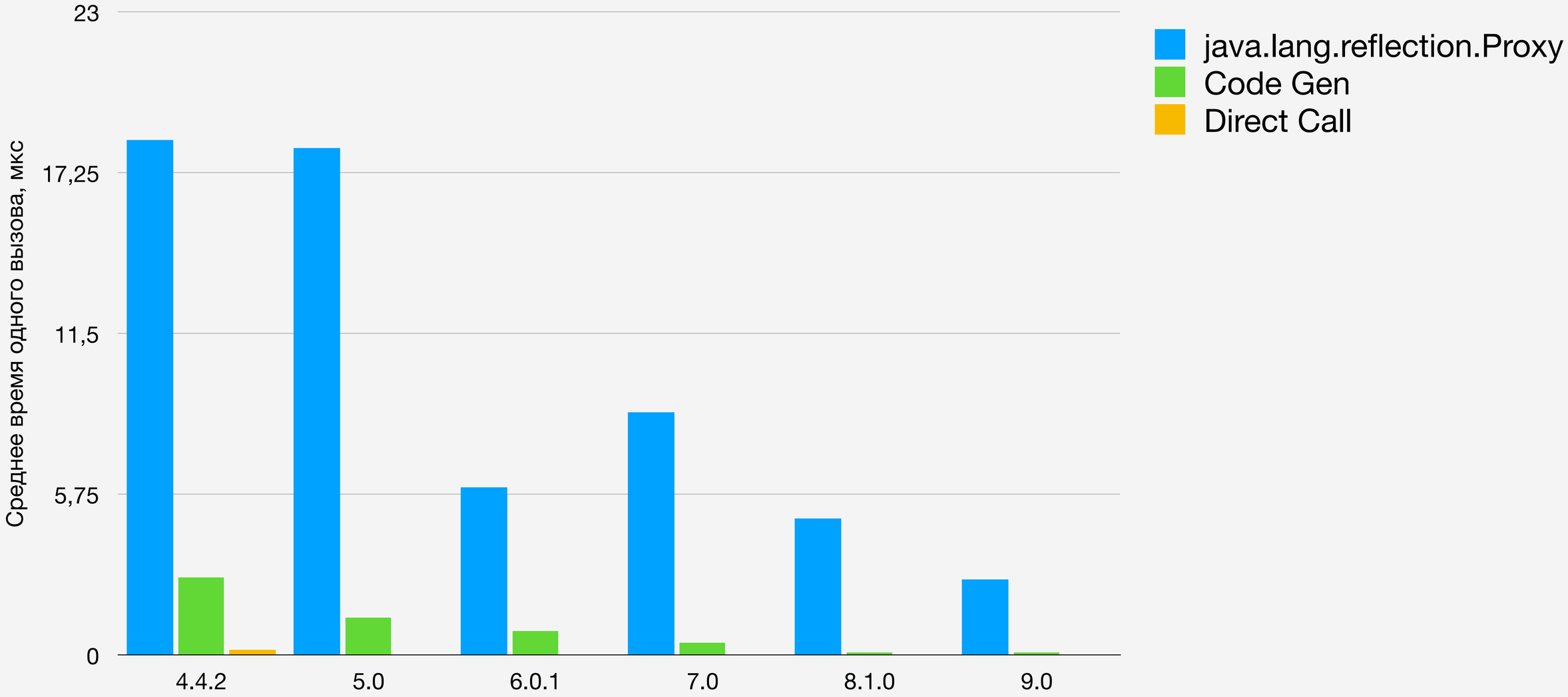
Без аргументов



3 примитива


23
17,25
11,5
5,75
0
Среднее время одного вызова, мкс
4.4.2
5.0
6.0.1
7.0
8.1.0
9.0
java.lang.reflection.Proxy
Code Gen
Direct Call

3 ссылочных типа



Готовые решения

dex-maker

fast-proxy

Спасибо за внимание