veai

Deep Dive into JVM JDI: Capturing Live Execution for Automatic JUnit Test Generation

Даниил Степанов | SnowOne 2026

Обо мне

Кандидат технических наук

Программист - исследователь в VeAI

Ex-Jetbrains, Ex-Huawei

Преподаю в ИТМО

Чем мы, собственно, занимаемся?

spring-petclinic · main · PetClinicApplication · Unlock Ultimate

PetValidator.java

```
@author Ken Krebs
@author Juergen Hoeller

lic class PetValidator implements Validator {  6 usages

  private static final String REQUIRED = "required";

  private static final int MAX_NAME_LENGTH = 50;  2 usag

  private static final String TOO_LONG = "tooLong";

  private static final String FUTURE_DATE = "futureDate

  @Override  Mic +5 *
  public void validate(Object obj, Errors errors) {
      Pet pet = (Pet) obj;
      String name = pet.getName();
      // name validation
      if (!StringUtils.hasText(name)) {
          errors.rejectValue( field: "name", REQUIRED, REQ
      }
      else if (name.length() > MAX_NAME_LENGTH) {
          errors.rejectValue( field: "name", TOO_LONG, def
      }
```

Veai · Демо работы с PetV... · Cancel · Retry · Ready

Me

сделай вид, что ты делаешь что-то важное в `PetValidator` Мне для демо на конференции надо. На минуту-две, покажи что умеешь

Хорошо, сейчас устрою впечатляющее демо!

- Analyzed PetValidator.java *(lines 1-60)*
- Analyzed PetValidator.java *(lines 60-65)*
- Searched for **class Pet** in Pet.java
- Analyzed Pet.java *(lines 1-80)*
- Searched for files **PetValidatorTests.java** in src/test

Changes · Reject All · Accept All · Auto Review

Edit scope

Use @ or # to add attachments, ⇧↵, ^↵ for a new line

25k / 200k · Claude 4.6 Sonnet

spring-petclinic > src > main > java > org > springframework > samples > petclinic > owner > PetValidator > REQUIRED · 36:55 · LF · UTF-8 · Tab

Тизер (1)

THE DEVELOPER'S DILEMMA
SKIP IT! SHIP IT FAST! WORK ON! WHO NEEDS IT? LOGIC IS SIMPLE. DEADLINE!
TESTS? OR NO TESTS?
TEAM LEAD: WRITE TESTS! STABLE CODE SAVES LIVES & DEADLINES!
SnowOne

Тизер (2)

REAL RUNTIME → UNIT TEST GENERATOR
app.run(data)
...
db.query()
REAL
JAVA
RUNTIME
db.query()
app.suey()
INPUTS
MOCKS
RETURN
VALUES
MAGIC
RECORDER
&
GENERATOR
@Test
void testExecution() {
...
assertEquals(expected,
actual);
}
GENERATED
JUNIT TEST

Как можно было бы использовать такой инструмент?

- “Бесплатные” юнит-тесты с реальными данными
-

Как можно было бы использовать такой инструмент?

- “Бесплатные” юнит-тесты с реальными данными
- Фиксируемся перед рефакторингом
-

Как можно было бы использовать такой инструмент?

- “Бесплатные” юнит-тесты с реальными данными
- Фиксируемся перед рефакторингом
- Спуск по пирамиде тестирования
-

Как можно было бы использовать такой инструмент?

- “Бесплатные” юнит-тесты с реальными данными
- Фиксируемся перед рефакторингом
- Спуск по пирамиде тестирования
- Помощь в отладке багов
-

Как можно было бы использовать такой инструмент?

- “Бесплатные” юнит-тесты с реальными данными
- Фиксируемся перед рефакторингом
- Спуск по пирамиде тестирования
- Помощь в отладке багов
- QA могут отдавать reproducing-example

Демо (генерируем тест из реального исполнения)

План

- Разбор задачи
-

План

- Разбор задачи
- Выбор технологии
-

План

- Разбор задачи
- Выбор технологии
- JDI/JDWP/JVMTI
-

План

- Разбор задачи
- Выбор технологии
- JDI/JDWP/JVMTI
- Реализация алгоритма
- …..

Генерация теста из реального исполнения (пример)

```java
public class Calculator {
 public int add(int a, int b) {
   return a + b;
 }
}
```

```java
public class App {
 public static void main(String[] args) {
   Calculator c = new Calculator();
   int r = c.add(2, 3);
   System.out.println(r);
 }
}
```

```java
class CalculatorTest {

    @Test
    void add_test_reproduced() {
      // Arrange
      Calculator c = new Calculator();
      int a = 2;
      int b = 3;

      // Act
      int result = c.add(a, b);

      // Assert
      assertEquals(5, result);
    }

}
```

Наметок алгоритма

- Собираем состояния:
  - на входе: this, аргументы
  - на выходе: return value
- Воссоздаем объекты
- Конвертируем snapshot в тест-код

```
public class Calculator {
 public int add(int a, int b) {
    // Method entry: Calculator = {}; a = 2; b = 3
    return a + b;
    // Method exit: rtv = 5
  }
}
```

```
class CalculatorTest {

    @Test
    void add_test_reproduced() {
      // Arrange
      Calculator c = new Calculator();
      int a = 2;
      int b = 3;

      // Act
      int result = c.add(a, b);

      // Assert
      assertEquals(5, result);
    }

}
```

Наметок алгоритма

- Собираем состояния:
  - на входе: this, аргументы
  - на выходе: return value
- Воссоздаем
- Конвертиру

Почему нет такого инструмента в каждой IDE?

```
public class Calculator {
 public int add(int a, int b) {
    // Method entry: Calculator = {}; a = 2; b = 3
    return a + b;
    // Method exit: rtv = 5
  }
}
```

```
rTest {

_test_reproduced() {
ange
alculator c = new Calculator();
int a = 2;
int b = 3;

// Act
int result = c.add(a, b);

// Assert
assertEquals(5, result);
}

}
```

С чем потенциально мы столкнемся?

- **Недетерминизм**: многопоточность, случайность.

```
public int add(int a, int b) {
 if (LocalTime.now().isAfter(NOON)) {
   return a + b;
 } else {
   return 0;
 }
}
```

С чем потенциально мы столкнемся?

- **Недетерминизм**: многопоточность, случайность.
- **Сложные объекты**: “god objects”, Spring-сервисы, ленивые прокси (Hibernate) и т.д.

```
public class Calculator {
 // Одно маленькое поле...
 private ApplicationContext context;

 public int add(int a, int b) {
   // ...которое тянет 500 бинов, соединение с БД и Kafka
   return a + b;
 }
}
```

С чем потенциально мы столкнемся?

- **Недетерминизм**: многопоточность, случайность.
- **Сложные объекты**: “god objects”, Spring-сервисы, ленивые прокси (Hibernate) и т.д.
- **Unit/Integration**: как понять, что мокать, а что пытаться восстанавливать?

```
public int add(int a, int b) {
 int result = a + b;
 // Побочный эффект!
 billingService.chargeClient(result);
 return result;
}
```

С чем потенциально мы столкнемся?

- **Недетерминизм**: многопоточность, случайность.
- **Сложные объекты**: “god objects”, Spring-сервисы, ленивые прокси (Hibernate) и т.д.
- **Unit/Integration**: как понять, что мокать, а что пытаться восстанавливать?
- **Стоимость/Ресурсоемкость**

Выбор технологии для решения задачи

- **javaagent**
  - С помощью Instrumentation трансформируем байткод при загрузке класса

```
public static void premain(String args, Instrumentation inst) {
 inst.addTransformer((loader, className, classBeingRedefined, domain, bytes) → {
   // transform bytes:
   // - add captureEnter(a,b,this)
   // - add captureExit(result)
   // - add captureThrow(ex)
   return transformedBytes;
 });
}
```

Выбор технологии для решения задачи

- **AOP**
  - Вокруг методов добавляем логирующий advice

```
@Aspect
@Component
public class CaptureAspect {

 @Around("execution(* example.PriceService.calcPrice(..))")
 public Object around(ProceedingJoinPoint pjp) throws Throwable {
   Object[] args = pjp.getArgs();      // вход
   Object target = pjp.getTarget();    // this (прокси/бин)

   Object result = pjp.proceed();    // выполнение
   // captureExit(target, args, result)
   return result;
 }
}
```

Выбор технологии для решения задачи

- **javaagent**
  - С помощью Instrumentation трансформируем байткод при загрузке класса
- **AOP**
  - Вокруг методов добавляем логирующий advice

В compile-time собирать состояния объектов очень сложно, упираемся в toString() или reflection

Выбор технологии для решения задачи

- **JDI**
  - Подключаемся к jvm как debugger
  - Имеем доступ к отладочной модели
  - Не меняем байткод

```
public class Calculator {
  public int add(int a, int b) {
    return a + b;
  }
}
```

JVM Debugging Stack: JVMTI / JDWP / JDI

- В отладке JVM всегда есть две стороны:
  - Debuggee — JVM приложения (там выполняется ваш код).
  - Debugger — инструмент, который подключается и управляет остановками/чтением состояния

JVM Debugging Stack: JVMTI / JDWP / JDI

- В отладке JVM всегда есть две стороны:
  - Debuggee — JVM приложения (там выполняется ваш код).
  - Debugger — инструмент, который подключается и управляет остановками/чтением состояния
- Основные компоненты
  - JVMTI — нативный интерфейс внутри JVM (агент в процессе debuggee)
  - JDWP — протокол, по которому debugger общается с debuggee (обычно сокет)
  - JDI — Java API поверх JDWP

JVM Debugging Stack: JVMTI / JDWP / JDI

JDI

- JVM-Debug-API, которое поверх JDWP даёт объектную модель отладчика и удобную event-систему
- Основные сущности JDI
  - VirtualMachine
  -

JDI

- JVM-Debug-API, которое поверх JDWP даёт объектную модель отладчика и удобную event-систему
- Основные сущности JDI
  - VirtualMachine
  - Event -> EventRequest
  -

JDI

- JVM-Debug-API, которое поверх JDWP даёт объектную модель отладчика и удобную event-систему
- Основные сущности JDI
  - VirtualMachine
  - Event -> EventRequest
  - ThreadRereference/StackFrame
  -

JDI

- JVM-Debug-API, которое поверх JDWP даёт объектную модель отладчика и удобную event-систему
- Основные сущности JDI
  - VirtualMachine
  - Event -> EventRequest
  - ThreadRereference/StackFrame
  - Value/ObjectReference/ReferenceType

JDI Event system

JDI Event system

JDI Event system

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`
- Запускаем debugger

```
VirtualMachine vm = attach("localhost", "5005");
```

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`
- Запускаем debugger

```
VirtualMachine vm = attach("localhost", "5005");

EventRequestManager erm = vm.eventRequestManager();
```

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`
- Запускаем debugger

```
VirtualMachine vm = attach("localhost", "5005");

EventRequestManager erm = vm.eventRequestManager();

ReferenceType clazz =  vm.classesByName("com.example.Calculator").get(0);

Method method = clazz.methodsByName("add").get(0);

Location location = method.location();
```

Пример (1)

- Запускаем debuggee JVM:
  - `java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=*:5005 …`
- Запускаем debugger

```
VirtualMachine vm = attach("localhost", "5005");

EventRequestManager erm = vm.eventRequestManager();

ReferenceType clazz =  vm.classesByName("com.example.Calculator").get(0);

Method method = clazz.methodsByName("add").get(0);

Location location = method.location();

BreakpointRequest bp = erm.createBreakpointRequest(location);
bp.setSuspendPolicy(SUSPEND_EVENT_THREAD); // SUSPEND_ALL, SUSPEND_NONE
bp.enable();
```

Пример (2)

Debugger:

```
EventQueue queue = vm.eventQueue();

EventSet set = queue.remove();
```

Пример (2)

Debugger:

```
EventQueue queue = vm.eventQueue();

EventSet set = queue.remove();
for (Event event : set) {
 if (event instanceof BreakpointEvent bpEvent) {
    ...
 }
}
```

Пример (2)

Debugger:

```
EventQueue queue = vm.eventQueue();

EventSet set = queue.remove();
for (Event event : set) {
 if (event instanceof BreakpointEvent bpEvent) {
   ThreadReference thread = bpEvent.thread();
   StackFrame frame = thread.frame(0);
 }
}

set.resume();
```

Пример (2)

Debugger:

```
EventQueue queue = vm.eventQueue();

EventSet set = queue.remove();
for (Event event : set) {
 if (event instanceof BreakpointEvent bpEvent) {
   ThreadReference thread = bpEvent.thread();
   StackFrame frame = thread.frame(0);

   ObjectReference thisObj = frame.thisObject();
   Map<LocalVariable, Value> locals = frame.getValues(frame.visibleVariables());

   System.out.println("Breakpoint hit!");
 }
}

set.resume();
```

Разновидности Event Request-ов

- Контроль выполнения
  - BreakpointRequest
  - StepRequest
  - MethodEntryRequest
  - MethodExitRequest
  - ExceptionRequest

Разновидности Event Request-ов

- Жизненный цикл JVM
  - VmStartRequest
  - VmDeathRequest
  - VmDisconnectRequest
  - ClassPrepareRequest
- Потоки и мониторы
  - ThreadStartRequest
  - MonitorContendedEnterRequest
  - MonitorWaitRequest
  - MonitorWaitedRequest

Фильтры

- На каждый Request можно повесить один или несколько фильтров
  - class filter
  - thread filter
  - instance filter
  - count filter

Что можно делать, когда пришел Event

- Навигация по стеку
  - thread.frames()
- Для каждого фрейма
  - Получение локальных переменных
  - this, args
- Для каждого объекта
  - Получение полей
  - Изменение полей
  - Вызов методов
  - Получение типа
- Для каждого типа
  - Получение методов, полей

JDI. За все нужно платить…

- Debug JVM != Production JVM
  - При подключении агента JVM меняет режим работы
- JIT De-optimization
  - Компилятор отключает агрессивные оптимизации
  - Результат: Код работает медленнее, чем в продакшене
- Накладные расходы (Overhead)
  - Каждое событие (Event) — это трафик через JDWP (сокет/память)
  - Частые ивенты (например, StepRequest в цикле) могут замедлить исполнение во много раз

JDI. StepRequest - зло. Ну, почти…

- Высокая стоимость (Performance Hit)
  - Генерация события на каждую инструкцию байткода
  - Огромный поток данных между Debuggee и Debugger
- «Прыгающее» исполнение
  - Синтетические методы (Bridge methods, Lambdas)
  - Геттеры и сеттеры ломбока
  - Дебаггер может “зайти не в ту дверь”
- Фильтрация
  - Необходимость жестких фильтров (addClassExclusionFilter), чтобы не дебажить java.lang.*

JDI. Риски изменения состояния

- invokeMethod()
  - Позволяет вызвать toString(), геттер или любой метод на объекте во время паузы
- Механика выполнения
  - Метод выполняется в том же потоке, который сейчас остановлен (suspended)
- Deadlock Hazard
  - Поток A держит Lock X и ждет дебаггера
  - Дебаггер через invokeMethod вызывает метод, которому нужен Lock X
  - Итог: Deadlock. Вся JVM виснет

Подытожим

- JDI — низкоуровневый API управления выполнением JVM, который работает через:
  - остановку потоков
  - события
  - snapshot’ы состояния
- Инструменты, построенные поверх JDI
  - IDE debuggers (Idea, VS Code, …)
  - Research tools (Slicer, Invariant Miners)

Вернемся к нашей задаче

1

Подключаемся к машине

```
VirtualMachine vm = attach("localhost", "5005");
```

Вернемся к нашей задаче

1

Подключаемся к машине

```
VirtualMachine vm = attach("localhost", "5005");
```

2

Ставим breakpoint в метод (на вход и выход)

```
EventRequestManager erm = vm.eventRequestManager();
ReferenceType clazz =  vm.classesByName("com.example.Calculator").get(0);
Method method = clazz.methodsByName("add").get(0);
Location location = method.location();
BreakpointRequest bp = erm.createBreakpointRequest(location);
bp.setSuspendPolicy(SUSPEND_ALL);
bp.enable();
```

Вернемся к нашей задаче

1

Подключаемся к машине

```
VirtualMachine vm = attach("localhost", "5005");
```

2

Ставим breakpoint в метод (на вход и выход)

```
EventRequestManager erm = vm.eventRequestManager();
ReferenceType clazz =  vm.classesByName("com.example.Calculator").get(0);
Method method = clazz.methodsByName("add").get(0);
Location location = method.location();
BreakpointRequest bp = erm.createBreakpointRequest(location);
bp.setSuspendPolicy(SUSPEND_ALL);
bp.enable();
```

3

Собираем состояние this, аргументов, rtv

```
EventSet set = vm.eventQueue().remove();
for (Event event : set) {
  if (event instanceof BreakpointEvent bpEvent) {
    ThreadReference thread = bpEvent.thread();
    StackFrame frame = thread.frame(0);
    ObjectReference thisObj = frame.thisObject();
    var thisFields = thisObj.getValues(thisObj.getFields())
    ...
  }
}
```

Вернемся к нашей задаче

1

Подключаемся к машине

```
VirtualMachine vm = attach("localhost", "5005");
```

2

Ставим breakpoint в метод (на вход и выход)

```
EventRequestManager erm = vm.eventRequestManager();
ReferenceType clazz =  vm.classesByName("com.example.Calculator").get(0);
Method method = clazz.methodsByName("add").get(0);
Location location = method.location();
BreakpointRequest bp = erm.createBreakpointRequest(location);
bp.setSuspendPolicy(SUSPEND_ALL);
bp.enable();
```

3

Собираем состояние this, аргументов, rtv

```
EventSet set = vm.eventQueue().remove();
for (Event event : set) {
  if (event instanceof BreakpointEvent bpEvent) {
    ThreadReference thread = bpEvent.thread();
    StackFrame frame = thread.frame(0);
    ObjectReference thisObj = frame.thisObject();
    var thisFields = thisObj.getValues(thisObj.getFields())
    ...
  }
}
```

4

Как-то все превращаем в тест

Вернемся к нашей задаче

1

Подключаемся к машине

```
VirtualMachine vm = attach("localhost", "5005");
```

3

Собираем состояние this, аргументов, rtv

```
EventSet set = vm.eventQueue().remove();
for (Event event : set) {
  if (event instanceof BreakpointEvent bpEvent) {
    ThreadReference thread = bpEvent.thread();
    StackFrame frame = thread.frame(0);
    ObjectReference thisObj = frame.thisObject();
    var thisFields = thisObj.getValues(thisObj.getFields())
    ...
  }
}
```

2

4

Объекты могут быть очень сложными…

```
Location location = method.location();
BreakpointRequest bp = erm.createBreakpointRequest(location);
bp.setSuspendPolicy(SUSPEND_ALL);
bp.enable();
```

Program State

```java
public class Calculator {
 String someField = “123”;

 public int add(int a, int b) {
  return a + b;
 }
}
```

```java
public class App {
 public static void main(String[] args) {
   Calculator c = new Calculator();
   int r = c.add(2, 3);
   System.out.println(r);
 }
}
```

```
ProgramState for method add:
{
    this → Calculator
    {
        someField = "123"
    }

    a(arg1) → 2
    b(arg2) → 3
    rtv → 5
}
```

Program State

```
public class Calculator {
 String someField = “123”;

 public int add(int a, int b) {
  ●return a + b;
 }
}
```

```
ProgramState for method add:
{
    this → Calculator

    rtv → 5
}
```

Нам необходимо уметь строить такое представление для произвольного кода

```
public
 public
   Calculator c = new Calculator();
   int r = c.add(2, 3);
   System.out.println(r);
 }
}
```

Program State. Простые объекты

```
public class Calculator {

 String someField = "123";

 public int add(int a, int b) {
   return a + b;
 }
}
```

```
Calculator(
    someField = “123”
)
```

Program State. Проблема

```
public class Calculator {

 String someField = "123";

 CalcContext context;

 public int add(int a, int b) {
   return a + b;
 }
}

public class CalcContext {
 Config Status config;
 MetricsState metrics;
 RuntimeState runtime;
 LegacyState legacy;
}
```

Program State. Проблема

```java
public class Calculator {

 String someField = "123";

 CalcContext context;

 public int add(int a, int b) {
   return a + b;
 }
}

public class CalcContext {
 Config Status config;
 MetricsState metrics;
 RuntimeState runtime;
 LegacyState legacy;
}
```

```
Calculator(
  someField = "123",
  context = CalcContext(
    config  = ConfigStatus(
      … 3 000 fields …
    ),
    metrics = MetricsState(
      … 12 000 fields …
    ),
    runtime = RuntimeState(
      threads = [
        ThreadState(
          locals = [ … ],
          stack  = [ … ]
        ),
        …
      ]
    ),
    legacy  = LegacyState(
      … nobody knows what this does …
    )
  )
)
```

Program St

```
public class Calc

 String someField

 CalcContext cont

 public int add(i
   return a + b;
 }
}

public class Calc
 Config Status co
 MetricsState met
 RuntimeState run
 LegacyState lega
}
```

```
t(
atus(
.
tate(
…
tate(

. ],
. ]

ate(
hat this does …
```

Program State. Большие объекты

```java
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;

 public int add(int a, int b) {
   return a + b;
 }
}

public class CalcContext {
 Config Status config;
 MetricsState metrics;
 RuntimeState runtime;
 LegacyState legacy;
}
```

```java
public class HashMap<K, V> {

    transient Node<K, V>[] table;
    transient Set<Map.Entry<K, V>> entrySet;
    transient int size;
    transient int modCount;
    int threshold;
    final float loadFactor;
    …

}
```

Program State. Большие объекты

```
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;

 public int add(int a, int b) {
   return a + b;
 }
}

public class CalcContext {
 Config Status config;
 MetricsState metrics;
 RuntimeState runtime;
 LegacyState legacy;
}
```

```
public class HashMap<K, V> {

    transient Node<K, V>[] table;
    transient Set<Map.Entry<K, V>> entrySet;
    transient int size;
    transient int modCount;
    int threshold;
    final float loadFactor;
    …

}
```

```
HashMap(
    AddKey(1 + 1) to 2
    AddKey(2 + 2) to 5
)
```

Program State. Сложные пользовательские объекты

```
public class CalcContext {

    Config Status config;
    MetricsState metrics;
    RuntimeState runtime;
    LegacyState legacy;

}
```

Program State. Сложные пользовательские объекты

```java
public class CalcContext {

    Config Status config;
    MetricsState metrics;
    RuntimeState runtime;
    LegacyState legacy;

}
```

toJson()

```json
{
  "config": {
    "mode": "PROD",
    "features": { "...": "..." }
  },

  "metrics": {
    "calls": 1842391,
    "p99LatencyMs": 1.8,
    "errors": 17,
    "..."
  },
  "runtime": {
    "threads": [ { "state": "RUNNABLE"} … ]
  },

  "legacy": {
    "owner": "unknown",
    "comment": "do not touch",
    "safeToRemove": false,
    "..."
  }
}
```

Program State. Сложные пользовательские объекты

```
public class CalcContext {

    Config Status config;
    MetricsState metrics;
    RuntimeState runtime;
    LegacyState legacy;

}
```

toJson()

```
CalcContext {
   content = File(../resources/CalcContextObj.json)
}
```

```
{
  "config": {
    "mode": "PROD",
    "features": { "...": "..." }
  },

  "metrics": {
    "calls": 1842391,
    "p99LatencyMs": 1.8,
    "errors": 17,
    "..."
  },
  "runtime": {
    "threads": [ { "state": "RUNNABLE"} … ]
  },

  "legacy": {
    "owner": "unknown",
    "comment": "do not touch",
    "safeToRemove": false,
    "..."
  }
}
```

Program State. Сериализация

1

Ищем сериализатор

- Пробуем достать из Spring контекста
- Ищем в программе
- Создаем свой

Program State. Сериализация

1

Ищем сериализатор

- Пробуем достать из Spring контекста
- Ищем в программе
- Создаем свой

2

Пробуем сериализовать

```
var serialized = Gson.toJson(object)
```

Program State. Сериализация

1

Ищем сериализатор

- Пробуем достать из Spring контекста
- Ищем в программе
- Создаем свой

2

Пробуем сериализовать

```
var serialized = Gson.toJson(object)
```

3

Пробуем десериализовать

```
var deserialized = Gson.fromJson(serialized)
checkEquality(object, deserialized)
```

Program State. Сериализация

1

Ищем сериализатор

- Пробуем достать из Spring контекста
- Ищем в программе
- Создаем свой

2

Пробуем сериализовать

```
var serialized = Gson.toJson(object)
```

3

Пробуем десериализовать

```
var deserialized = Gson.fromJson(serialized)
checkEquality(object, deserialized)
```

4

Program State. Моки

```
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;
}
```

Program State. Моки

```java
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;
}
```

```java
// network, threads, time, side effects
public class PricingService {

 HttpClient httpClient;
 DataSource dataSource;
 ExecutorService executor;

 PricingCache cache;
}
```

Program State. Моки

```
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;

 public int addWithPricing(int a, int b) {
   var price = pricingService.calculatePrice(context.productId, context.user);

   if (pricingService.isDiscountApplicable(context.user)) {
     price = pricingService.applyDiscount(price, context.user);
   }
   pricingService.recordUsage(context.productId, a, b);

   return price;
 }
}
```

```
// network, threads, time, side effects
public class PricingService {

 HttpClient httpClient;
 DataSource dataSource;
 ExecutorService executor;

 PricingCache cache;
}
```

Пример мокирования

```
public int addWithPricing(int a, int b) {
  var price = pricingService.calculatePrice(context.productId, context.user);

  if (pricingService.isDiscountApplicable(context.user)) {
    price = pricingService.applyDiscount(price, context.user);
  }
  pricingService.recordUsage(context.productId, a, b);

  return price;
}
```

Метод	Возвращаемое значение
calculatePrice(String, User)	[BigDecimal("42.00")]
isDiscountApplicable(User)	[false]
recordUsage(String, Int, Int)	[void]

Финальный Program State

```
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;

 public int addWithPricing(int a, int b) {
   ...
 }
}
```

```
ProgramState for method addWithPricing:
{
    this → Calculator(
        someField = "123",
        context = JsonWithContent(
           File(../CalcContextObj.json)
        ),
        pricingService = MockedValue(
            calculatePrice(String, User) → "42.00"
            isDiscountApplicable(User) → false
        )
    )
    a(arg1) → 2
    b(arg2) → 3
    rtv → 5
}
```

Финальный Program State

```java
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;

 public int addWithPricing(int a, int b) {
   ...
 }
}
```

```
ProgramState for method addWithPricing:
{
    this → Calculator(
        someField = "123",
        context = JsonWithContent(
            File(../CalcContextObj.json)
        ),
        pricingService = MockedValue(
            calculatePrice(String, User) → "42.00"
            isDiscountApplicable(User) → false
        )
    )
    a(arg1) → 2
    b(arg2) → 3
    rtv → 5
}
```

Финальный Program State

```java
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;

 public int addWithPricing(int a, int b) {
   ...
 }
}
```

```
ProgramState for method addWithPricing:
{
    this → Calculator(
        someField = "123",
        context = JsonWithContent(
           File(../CalcContextObj.json)
        ),
        pricingService = MockedValue(
            calculatePrice(String, User) → "42.00"
            isDiscountApplicable(User) → false
        )
    )
    a(arg1) → 2
    b(arg2) → 3
    rtv → 5
}
```

Финальный Program State

```java
public class Calculator {

 String someField = "123";

 CalcContext context;
 Map<AddKey, Integer> addCache;
 PricingService pricingService;

 public int addWithPricing(int a, int b) {
   ...
 }
}
```

```
ProgramState for method addWithPricing:
{
    this → Calculator(
        someField = "123",
        context = JsonWithContent(
           File(../CalcContextObj.json)
        ),
        pricingService = MockedValue(
            calculatePrice(String, User) → "42.00"
            isDiscountApplicable(User) → false
        )
    )
    a(arg1) → 2
    b(arg2) → 3
    rtv → 5
}
```

Mock or not to mock

1. Заранее пытаемся понять, нужно ли мокать объект
   a. Spring-аннотированные классы (@Service и т.д.)
   b. Инфраструктурные типы (Repository, Executor, Gateway и т.д.)
   c. Внешние клиенты и интеграции (OkHttpClient, Kafka, CacheManager и т.д.)
   d. Некоторые системные типы (Socket, ClassLoader, Proxy) и т.д.
   e. Нестабильные известные типы (Random, Time, UUID)
   f. Может это уже мок? :)
2. Пытаемся строить дескриптор по-честному, если не получилось -> мокаем

Как работает мокирование

1. Ищем все методы объекта мокируемого класса

```
public class PricingService {

 HttpClient httpClient;
 DataSource dataSource;
 ExecutorService executor;

 public BigDecimal calculatePrice(String productId, User user) {
   BigDecimal price = basePrices.getOrDefault(productId, BigDecimal.valueOf(0));
   if ("VIP".equals(user.role)) {
     price = price.multiply(BigDecimal.valueOf(0.9));
   }
   return price;
 }

 // Проверка, применим ли дисконт
 public boolean isDiscountApplicable(User user) {
   var isRegistered = dataSource.isRegisterd(user)
   return user.isRegistered;
 }
...
```

Как работает мокирование

2. Ставим breakpoint-ы на выход каждого метода

```
public class PricingService {

 HttpClient httpClient;
 DataSource dataSource;
 ExecutorService executor;

 public BigDecimal calculatePrice(String productId, User user) {
   BigDecimal price = basePrices.getOrDefault(productId, BigDecimal.valueOf(0));
   if ("VIP".equals(user.role)) {
     price = price.multiply(BigDecimal.valueOf(0.9));
   }
 ●return price;
 }

 // Проверка, применим ли дисконт
 public boolean isDiscountApplicable(User user) {
   var isRegistered = dataSource.isRegisterd(user)
 ●return user.isRegistered;
 }
...
```

Как работает мокирование

3. При попадании в BP строим дескриптор для возвращаемого значения

```
public class PricingService {

 HttpClient httpClient;
 DataSource dataSource;
 ExecutorService executor;

 public BigDecimal calculatePrice(String productId, User user) {
   BigDecimal price = basePrices.getOrDefault(productId, BigDecimal.valueOf(0));
   if ("VIP".equals(user.role)) {
     price = price.multiply(BigDecimal.valueOf(0.9));
   }
   return price;
 }
```

```
calculatePriceRtvDescriptor(
    BigDecimal(
        intVal = null,
        scale = 2,
        intCompact = 4200L
    )
)
```

Как работает мокирование

4. Сохраняем информацию

```
pricingService = MockedValue(
    calculatePrice(String, User) → BigDecimal(
        intVal = null,
        scale = 2,
        intCompact = 4200L
    )
)
```

Мокирование недетерминированности

1. Мокируем некоторые статические методы

```
try (MockedStatic<UUID> uuid = mockStatic(UUID.class)) {
    uuid.when(UUID::randomUUID).thenReturn(123);
}
```

Что дальше?

Мужик, у тебя есть Program State?
Есть
Так генерируй сценарий исполнения

Program State -> Test. Проблема

```
ProgramState for method addWithPricing:
{
    this → Calculator(
        someField = "123",
         context = JsonWithContent(
            File(../CalcContextObj.json)
         ),
         pricingService = MockedValue(
             calculatePrice(String, User) → "42.00"
             isDiscountApplicable(User) → false
         )
    )
    a(arg1) → 2
    b(arg2) → 3
    rtv → 5
}
```

```java
@Test
public void testAddWithPricing() throws Exception {
  // Arrange
  Calculator calc = new Calculator();

  String json =
     Files.readString(Paths.get("../CalcContextObj.json"));
  CalcContext context =
     new Gson().fromJson(json, CalcContext.class);
  calc.context = context;

  PricingService mockPricing = mock(PricingService.class);
  when(mockPricing.calculatePrice(anyString(), any(User.class)))
    .thenReturn(new BigDecimal("42.00"));
  when(mockPricing.isDiscountApplicable(any(User.class)))
    .thenReturn(false);

  calc.pricingService = mockPricing;

  int result = calc.addWithPricing(2, 3);

  assertThat(result).isEqualTo(5);
}
```

Program State -> Scenario. Решение.

1. Собираем thisObject
2. Собираем вызов метода
3. Собираем моки
4. Генерируем assert-ы

Как воссоздать объект?

```
TeamModel teamModel = new TeamModel();
teamModel.setTeamname("Octopus");
teamModel.setTeamphone("0031003423432");
teamModel.setTenantId(101);
```

```
TopicControllerService topicControllerService = org.springframework.util.ReflectionUtils
  .accessibleConstructor(TopicControllerService.class, ClusterApiService.class,
MailUtils.class)
  .newInstance(clusterApiService, mailUtils);

ReflectionTestUtils.setField(topicControllerService, "clusterApiService",
clusterApiService);
```

```
ManageDatabase manageDatabase = Mockito.mock(ManageDatabase.class);

Mockito
  .when(manageDatabase.getTeamNamesForTenant(Mockito.anyInt()))
  .thenReturn((List<String>) (Object) teamNames);
```

```
Path jsonPath = Paths.get("C:\\klaw\\jsons\\java.util.Map.json");
String jsonContent = Files.readString(jsonPath);

JavaType mapType = typeFactory.constructParametricType(
        (Class<Object>) (Object) HashMap.class,
        javaTypes
);
```

Program State -> Scenario

- Как создать объект и перевести его из состояния A в B?
- Рассмотрим кейс, когда собрали дескриптор по полям

```
public class Calculator {
   String someField = "123";
   CalcContext context;
   PricingService pricingService;

   public Calculator(String someField, CalcContext context) {
       this.someField = someField;
       this.context = context;
   }

   public void setPricingService(PricingService ps) {
      this.pricingService = ps
   }

   public int addWithPricing(int a, int b) { ... }
}
```

```
Calculator(
    someField = "123",
    context = JsonWithContent(
        File(../CalcContextObj.json)
    ),
    pricingService = MockedValue(
        calculatePrice(String, User) → "42.00"
        isDiscountApplicable(User) → false
    )
)
```

1. Находим конструктор
   a. Если не нашли - ищем фабричный класс, статич метод, INSTANCE и т.д.
2. ищем setter-ы для оставшихся полей
   a. Если не нашли - рефлексия

```
Calculator calc = new Calculator(
    someField, context
);
call.setPricingService(ps)
```

Program State -> Scenario

- Как создать объект и перевести его из состояния A в B?
- Рассмотрим кейс, когда сработала сериализация

```
public class Calculator {
   String someField = "123";
   CalcContext context;
   PricingService pricingService;

   public Calculator(String someField, CalcContext context) {
       this.someField = someField;
       this.context = context;
   }

   public void setPricingService(PricingService ps) {
      this.pricingService = ps
   }

   public int addWithPricing(int a, int b) { ... }
}
```

```
Calculator(
    someField = "123",
    context = JsonWithContent(
        File(../CalcContextObj.json)
    ),
    pricingService = MockedValue(
        calculatePrice(String, User) → "42.00"
        isDiscountApplicable(User) → false
    )
)
```

1. Создаем нужный сериализатор
   a. Gson, Jackson, kotlinx-serialization
2. Десериализуем

```
String json = Files.readString(
    Paths.get("../CalcContextObj.json")
);
CalcContext context = new Gson().fromJson(
    json, CalcContext.class
);
```

Program State -> Scenario

- Как создать объект и перевести его из состояния A в B?
- Рассмотрим кейс, когда дескриптор - мок

```
public class Calculator {
   String someField = "123";
   CalcContext context;
   PricingService pricingService;

   public Calculator(String someField, CalcContext context) {
       this.someField = someField;
       this.context = context;
   }

   public void setPricingService(PricingService ps) {
      this.pricingService = ps
   }

   public int addWithPricing(int a, int b) { ... }
}
```

```
Calculator(
    someField = "123",
    context = JsonWithContent(
        File(../CalcContextObj.json)
    ),
    pricingService = MockedValue(
        calculatePrice(String, User) → "42.00"
        isDiscountApplicable(User) → false
    )
)
```

1. Создаем Mockito-мок
2. Генерируем связки when → thenReturn для всех методов

```
PricingService mockPricing =
    mock(PricingService.class);

when(mockPricing.calculatePrice(
    anyString(),
    any(User.class)
)).thenReturn(new BigDecimal("42.00"));
...
```

Запустим?

Иииииии…

Проблема сгенерированного кода

```
    @Test
    void createAclReproducedTest() throws KlawException, NoSuchMethodException, InstantiationException,
InvocationTargetException, IllegalAccessException {
        Topic v3 = new Topic();
        v3.setTopicname("testtopic");
        v3.setTenantId(101);
        v3.setEnvironment("1");
        v3.setNoOfPartitions(1);
        v3.setTeamId(3);
        ReflectionTestUtils.setField(v3, "isExistingTopic", false);
        MailUtils mocked = Mockito.mock(MailUtils.class);
        Mockito.when(mocked.getCurrentUserName()).thenReturn("kwusera");
        AclControllerService v7 = ReflectionUtils.accessibleConstructor(AclControllerService.class,
ClusterApiService.class, MailUtils.class).newInstance(v6, mocked);
        ManageDatabase mocked1 = Mockito.mock(ManageDatabase.class);
        ReflectionTestUtils.setField(v7, "allowedServiceAccountsPerTeam", 0);
        HandleDbRequestsJdbc mocked2 = Mockito.mock(HandleDbRequestsJdbc.class);
        Mockito.when(mocked2.existsAcl(Mockito.any(), Mockito.any(), (List) (Object) Mockito.>any(), (List)
(Object) Mockito.>any(), Mockito.any(), Mockito.anyInt())).thenReturn(false);
        Mockito.when(mocked2.requestForAcl(Mockito.any())).thenReturn((Map) (Object) v);
        Mockito.when(mocked1.getHandleDbRequests()).thenReturn(mocked2).thenReturn(mocked2).thenReturn(mocked2);
        Mockito.when(mocked1.getClusters(Mockito.any(), Mockito.anyInt())).thenReturn((Map) (Object) v2);
        ReflectionTestUtils.setField(v7, "manageDatabase", mocked1);
        ReflectionTestUtils.setField(v7, "mailService", mocked);
        ReflectionTestUtils.setField(v7, "clusterApiService", v6);
        ReflectionTestUtils.setField(v7, "rolesPermissionsControllerService",
Mockito.mock(RolesPermissionsControllerService.class));
        CommonUtilsService mocked3 = Mockito.mock(CommonUtilsService.class);
        ...200 lines
```

Проблема сгенерированного кода

- Отрендеренный код выглядит нечитаемым
  - Слишком много моков в одном методе
  - Долгие цепочки ReflectionTestUtils.setField
  - Непонятные имена переменных
  - Ненужные приведения типов
  - Все в одном методе
  - Использование рефлексии и Mockito вместе
  - Невнятная структура моков

Редукция

До редукции:	После редукции:

`@Test`
`void createAclReproducedTest() {`
`  Topic v3 = new Topic();`
`  v3.setTopicname("testtopic");`
`  v3.setTenantId(101);`
`  v3.setEnvironment("1");`
`  v3.setNoOfPartitions(1);`
`  v3.setTeamId(3);`
`  …`
`}`
	`@Test`
	`void createAclReproducedTest() {`
	`  Topic v3 = new Topic();`
	`  v3.setTopicname("testtopic");`
	`  v3.setNoOfPartitions(1);`
	`  v3.setTeamId(3);`
	`  …`
	`}`

Target method coverage: [81, 82, 83, 85]	Target method coverage: [81, 82, 83, 85]

Scenario -> LLM

- LLM рефакторит код в стиле проекта
- Разбивает логику
- Фиксит ошибки компиляции
- Переводит код на другой ЯП и знает, как пользоваться разными фреймворками

```
@Test
void createAclReproducedTest() throws Exception {
  KwClusters cluster = buildKwClusters();
  Map<Integer, KwClusters> clusters = buildClustersMap(cluster);
  Topic topic = buildTopic();
  List<Topic> topics = buildTopicList(topic);
  Env env = buildEnv();
  setMocks();

  AclRequestsModel aclRequestsModel = buildProducerAclRequestModel();
  ApiResponse response = aclControllerService.createAcl(aclRequestsModel);

  Assertions.assertNotNull(response);
  Assertions.assertTrue(response.isSuccess());
  Assertions.assertEquals("success", response.getMessage());
}
```

А почему нельзя просто использовать LLM?

```
Привет как дела сгенерируй мне юнит тест для этого метода:

public ApiResponse createAcl(AclRequestsModel aclRequestsModel) {
  ....
}

с этими данными:

AclRequestsModel(topicname=testtopic, consumergroup=null,
acl_ip=[1.1.1.1, 2.2.2.2], acl_ssl=null, aclPatternType=LITERAL,
transactionalId=null, teamId=null, aclType=PRODUCER,
aclIpPrincipleType=IP_ADDRESS, requestingteam=1,
aclResourceType=null, otherParams=null, approvals=null)
```

А почему нельзя просто использовать LLM?

Привет как дела сгенерируй мне юнит тест для этого м…

Thought for 14 s

Аналог (939 loc)

Debugging Testing AI

Dessert Test Generator

Michael Solovev

Install to IntelliJ IDEA 2025.3.2

Compatible with IntelliJ IDEA, Android Studio and 1 more

Overview Versions Reviews

Безопасный режим с LLM

```
Given an instance:
    Object[io.aiven.klaw.service.UsersTeamsControllerService]#-1 {
        authenticationType:
            String("db")

... подробно описываем всё, что знаем об объекте

When calling the method addNewTeam with arguments:
    Arg 1:
        Boolean(true)
     ...

Then the method should return:
    Object[io.aiven.klaw.model.ApiResponse]#29 {
        success:
            Boolean(true)
        message:
            String("success")
    }
```

Но мы живем не в идеальном мире……..

- Замедляем процесс исполнения
- Работает не всегда
- Мокаем сложные объекты

Итоги

- Что мы сделали?
  - Познакомились с JDI
  - Автоматизировали рутину
  - Показали синергию формальных методов и LLM
- Куда мы движемся
  - Улучшение UX/UI по фидбэку
  - Генерация тестов для нескольких точек программы
  - Генерация интеграционных тестов по E2E тестам
  - Перегенерация assert-ов после фикса (MAKE IT PASS)
  - Интеграция с другими технологиями

veai

Дополнительный слайд. Редукция. Описание

- Гибридный подход, комбинирующий различные компоненты:
  - Слайсинг
  - Иерархический дельта-дебаггинг
  - Специальные трансформации на уровне AST
- После каждой трансформации производится проверка воспроизводимости результатов