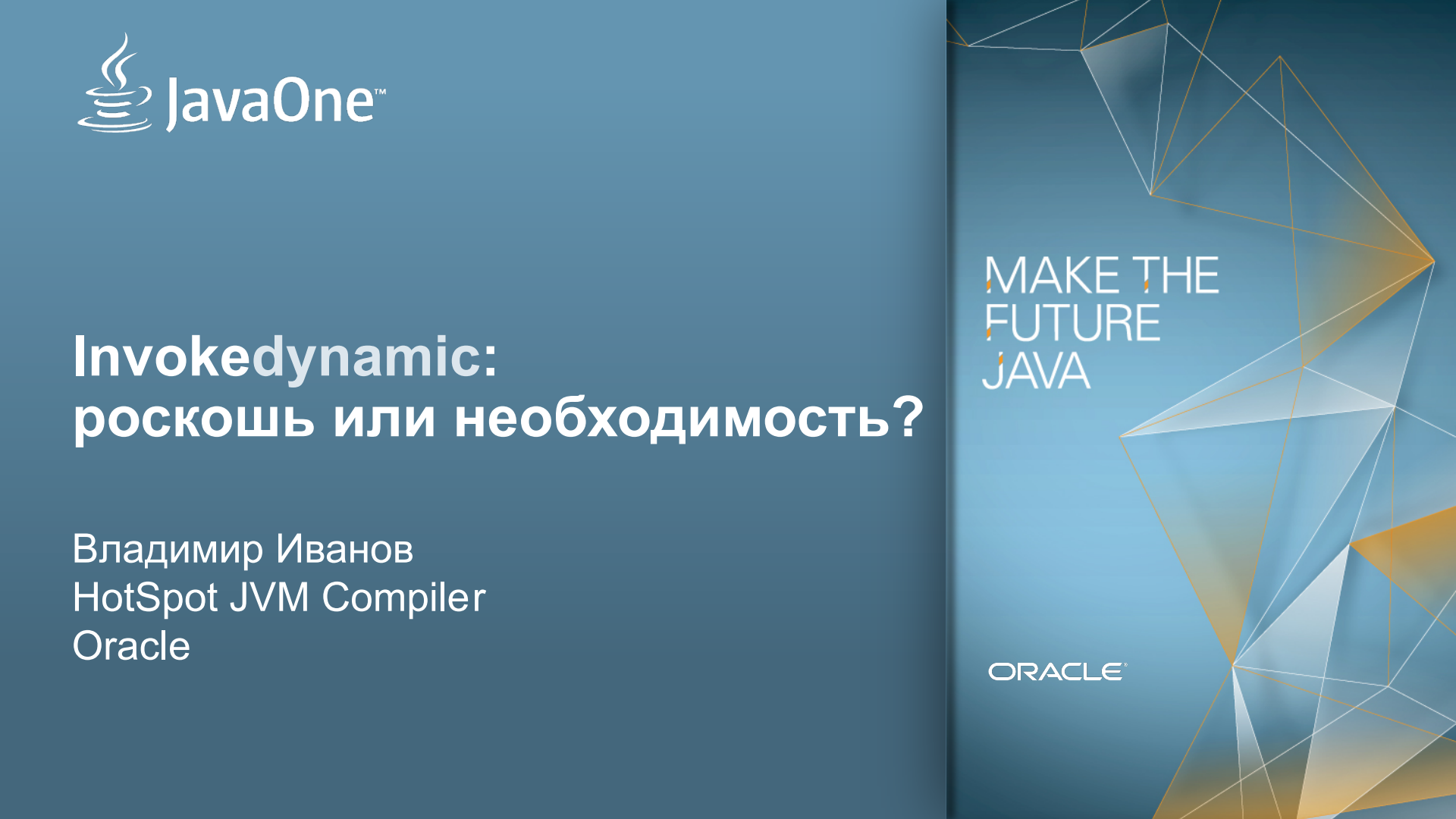


Invokedynamic: роскошь или необходимость?

Владимир Иванов
HotSpot JVM Compiler
Oracle

Что Вас ждет в ближайший час

- **in**voke**dy**namic (**indy**)
  - invoke, простите, что?!
- Method handles (aka method pointers)
  - они-то тут причем?..
- Детали-детали-детали
  - понемногу, но обо всем

Немного истории…

JSR 292: Supporting Dynamically Typed Languages on the Java Platform

- 2011, July 28: Released as part of Java 7
- 2010: API refinement (e.g., BootstrapMethods)
- 2009: API refinement (e.g., CONSTANT_MethodHandle)
- 2008: API with Method Handles (Early Draft Review)
- 2007: Expert Group reboot
- 2006: JSR 292 Expert Group formed
- 2005: Initial design sketch

Что такое invokedynamic?

"In summary, invokedynamic…

- is a natural general purpose primitive
  - Not tied to semantics of a specific programming language
  - Flexible building block for a variety of method invocation semantics
- enables relatively simple and efficient method dispatch."

Gilad Bracha

Java Language Architect,
Sun Microsystems, JAOO, 2005

Что такое indy?

“In summary, invokedynamic…

- is a natural general purpose primitive
  - Not tied to semantics of a specific programming language
  - Flexible building block for a variety of method invocation semantics
- enables relatively simple and efficient method dispatch.”

Gilad Bracha

Java Language Architect,
Sun Microsystems, JAOO, 2005

JSR 292

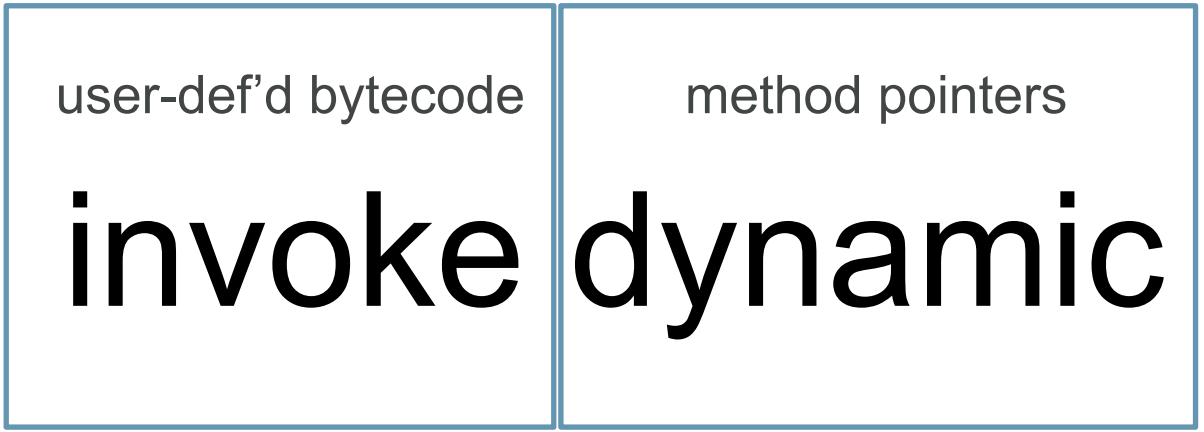
Нововведения

- Инструкция с программируемым поведением
  - полная свобода в выборе поведения JVM
- Быстрые «указатели на методы» и адаптеры
- Оптимизируется не хуже чем обычный Java код
- Избежать модификаций в будущем

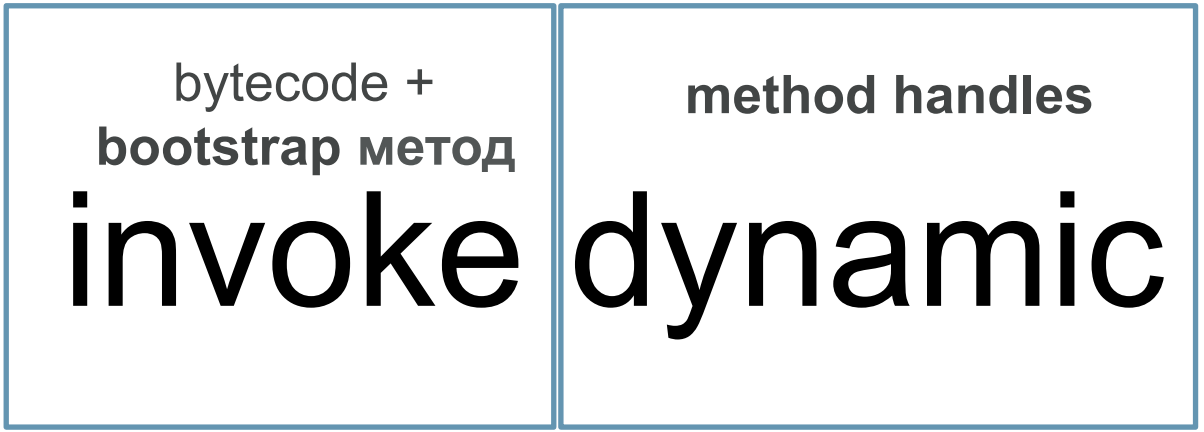
Ключевые нововведения

- новая инструкция: invokedynamic.
  - линкуется под контролем пользователя
  - видимый пользователю объект: java.lang.invoke.CallSite
  - может быть слинкована и потом перелинкована
- новый «атом» поведения: method handle
  - call site содержит ссылку на method handle
  - Method handle – указатель на метод для JVM
  - (или, каждый MH реализует интерфейс с одним методом)

Что такое indy?



Что такое indy?



Архитектура

- Байткод порождается Java компиляторами или «на лету».

Архитектура

- Байткод порождается Java компиляторами или «на лету».

- Динамический call site создается **для каждой** инструкции invokedynamic.

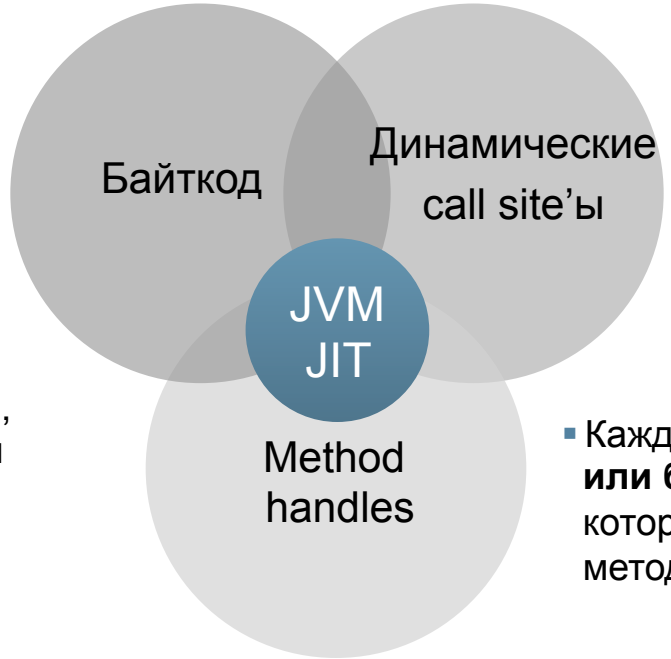
Архитектура

- Байткод порождается Java компиляторами или «на лету».

- Динамический call site создается **для каждой** инструкции invokedynamic.
- Каждый call site связан с **одним или более** method handle'ом, который указывает на некоторый метод в байткоде.

Архитектура

- Байткод порождается Java компиляторами или «на лету».



- Динамический call site создается **для каждой** инструкции invokedynamic.
- При необходимости, JVM, прозрачно для приложения, оптимизирует в машинный код.
- Каждый call site связан с **одним или более** method handle'ом, который указывает на некоторый метод в байткоде.

invoke*

bytecode + bootstrap метод

Invokedynamic на уровне Java байткоде

Изменения в структуре .class файлов

- Новая инструцкия
  - invokedynamic (0xBA)
- Новые типы элементов в constant pool’е
  - CONSTANT_InvokeDynamic_info
  - CONSTANT_MethodHandle_info
  - CONSTANT_MethodType_info
- улучшенный LDC
- Атрибут BootstrapMethods

Invokedynamic

Сравнение с другими invoke-инструкциями

invokestatic	invokespecial	invokevirtual	invokeinterface	invokedynamic
no receiver	no receiver	receiver class	receiver interface	**no receiver**
no dispatch	no dispatch	single dispatch	single dispatch	**custom dispatch**

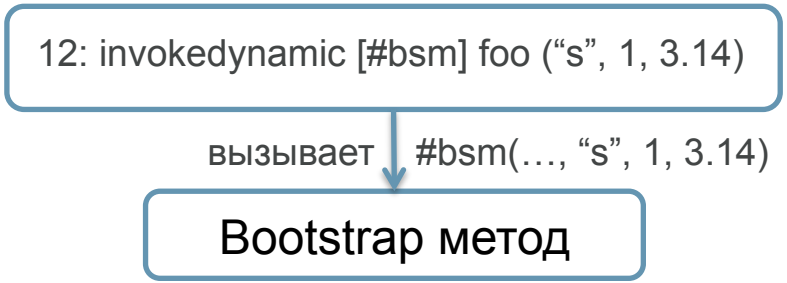
Invokedynamic: логика работы

1й вызов

12: invokedynamic [#bsm] foo (“s”, 1, 3.14)

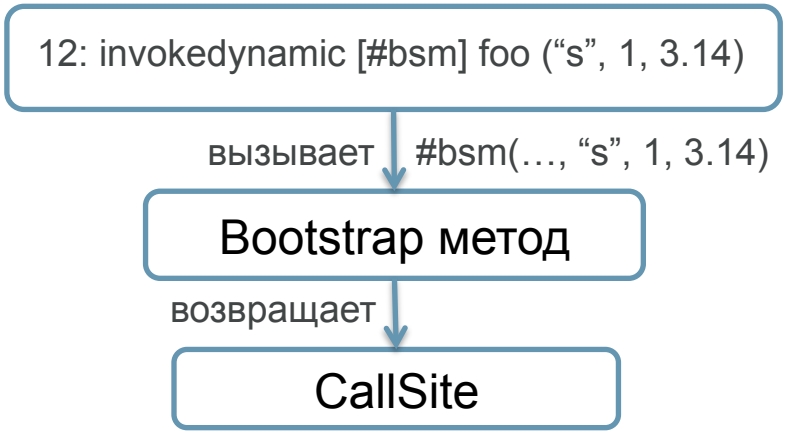
Invokedynamic: логика работы

1й вызов



Invokedynamic: логика работы

1й вызов

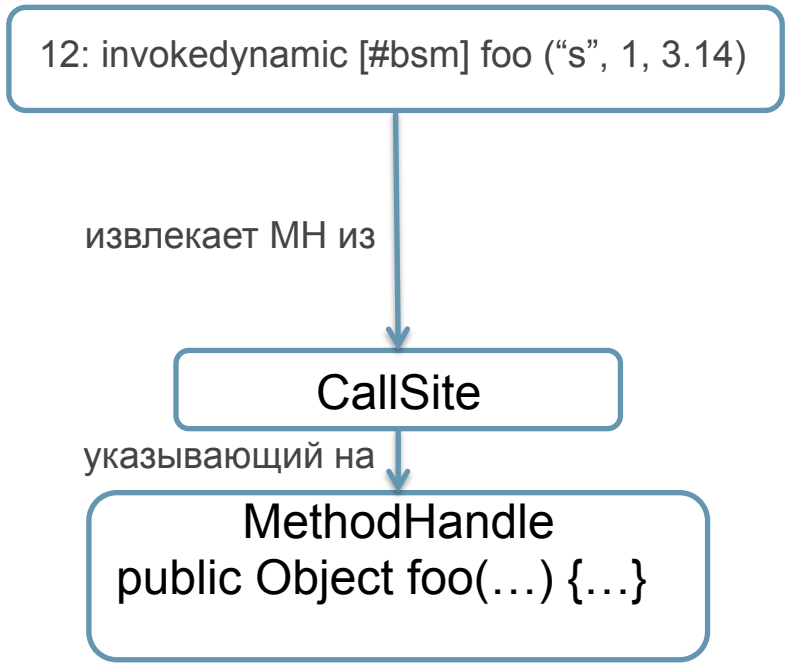


Invokedynamic: логика работы

1й вызов

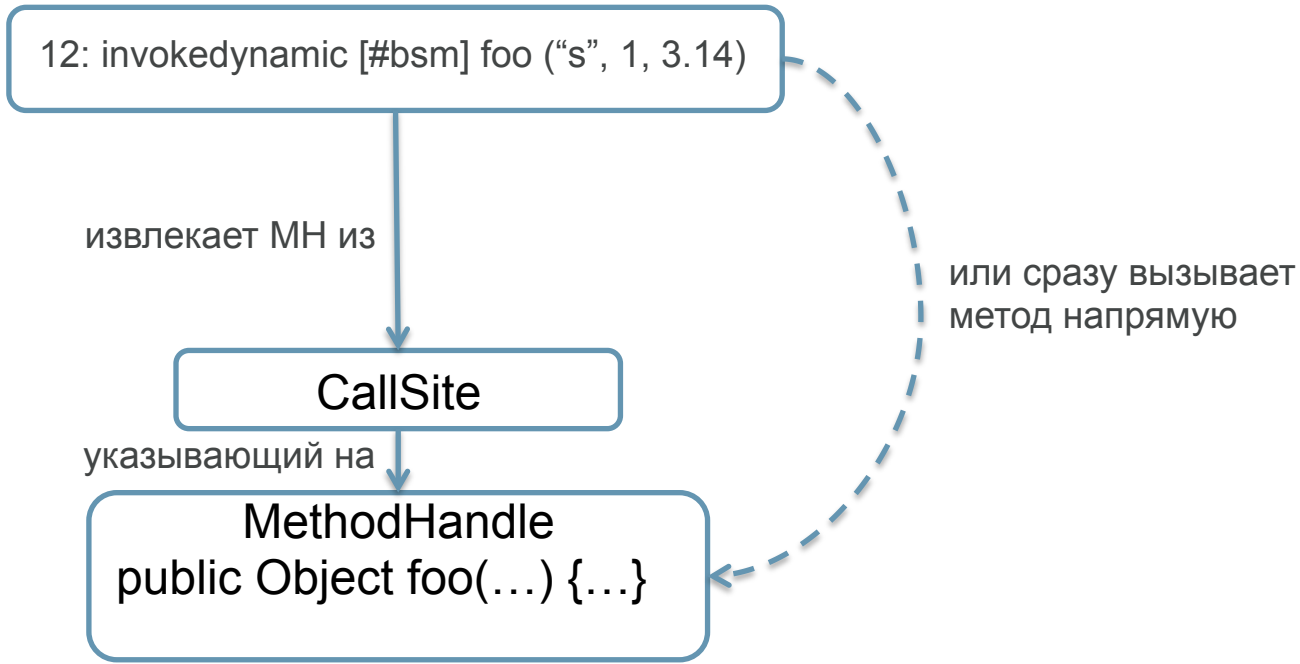
Invokedynamic: логика работы

2й и последующие вызовы



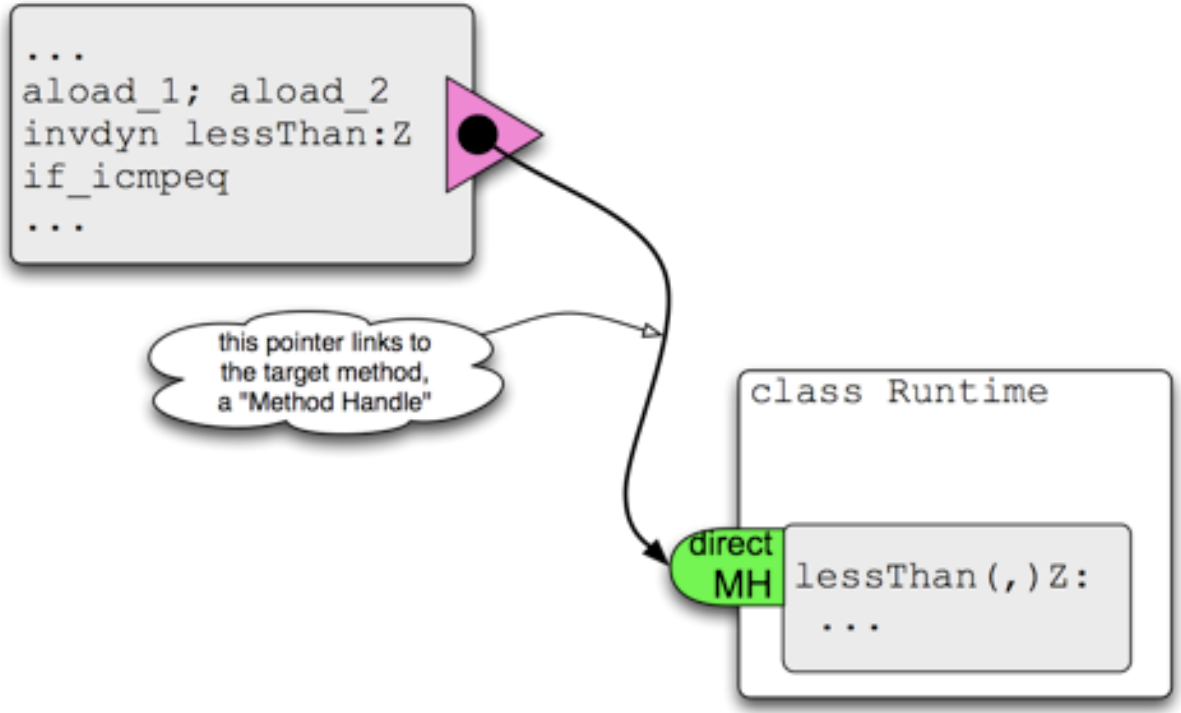
Invokedynamic: логика работы

2й и последующие вызовы



Invokedynamic: логика работы

2й и последующие вызовы



Bootstrap метод

- Изначально, каждый call site не слинкован
- Call site должен быть слинкован перед первым вызовом
  - может быть «лениво»
- Линкуется посредством вызова bootstrap метода
  - которому передается статическая информация о call site’е
    - контекст, имя метода, сигнатура метода, дополнительные параметры
  - возвращает объект типа CallSite, содержащий ссылку на метод
- каждая invokedynamic инструцкия статически определяет свой bootstrap метод (как ссылку в constant pool)

java.lang.invoke.CallSite

- Объект, описывающий «место» вызова в коде
- Хранит ссылку на метод («цель»), который должен вызываться
- Линкуется с invokedynamic инструцией через bootstrap метод
- Может быть нескольких типов

CallSite

Типы

- ConstantCallSite
  - «цель» (method handle) после создания меняться не может
- MutableCallSite
  - как обычное поле, хранящее ссылку на метод
  - делегирует каждый вызов текущему MH
- VolatileCallSite
  - как volatile поле
  - изменения видны сразу
  - … за счет потенциального ухудшения производительности

Вызов через invokedynamic

- Вызов метода через инструкцию invokedynamic быстр
  - всегда «точный»
    - типы параметров полностью совпадают
    - call site’ы строго типизированы
  - нет необходимости в проверке типа при вызове
    - VM гарантирует корректность
  - код, на который указывает method handle часто может быть заинлайнен

invokedynamic

Пример: байткод

```
invokedynamic baz:(LSomeClass;ID)V [#bsm2, 1234.5]
```

invokedynamic

Пример: байткод

```
invokedynamic baz:(LSomeClass;ID)V [#bsm2, 1234.5]

static CallSite bsm2(Lookup lookup, String name, MethodType type, Object... arg) {
    MethodHandle mh = lookup.findVirtual(SomeClass.class, name, type);
    return new ConstantCallSite(mh);
}
```

invokedynamic

Пример: байткод

```
invokedynamic baz:(LSomeClass;ID)V [#bsm2, 1234.5]

static CallSite bsm2(Lookup lookup, String name, MethodType type, Object... arg) {
    MethodHandle mh = lookup.findVirtual(SomeClass.class, name, type);
    return new ConstantCallSite(mh);
}

== invokevirtual SomeClass.baz:(ID)V
```

*dynamic

Method Handles

Method Handles

"The most fundamental change to Java since it's inception."

Charles Oliver Nutter
JRuby Lead Developer

Method Handles

Что это?

- Краеугольный камень для JSR 292
- Основа для инструкции invokedynamic
- Ключевая функциональность для других проектов

Method Handles

Что это?

- Указатели на методы/поля/элементы массива
- Манипуляция аргументами
- Управляющая логика
- Должно быть оптимизируемо JVM
  - это очень важно!

Invoke инструкции

До JSR292

- invoke*

```
invokevirtual      java/io/PrintStream.println:()V
invokeinterface    java/util/List.add:(Ljava/lang/Object;)Z
invokestatic       java/lang/System.currentTimeMillis:()J
invokespecial      java/lang/Object.<init>:()V
```

Invoke инструкции

И не только invoke*

- invoke*

  invokevirtual java/io/PrintStream.println:()V

  invokeinterface java/util/List.add:(Ljava/lang/Object;)Z

  invokestatic java/lang/System.currentTimeMillis:()J

  invokespecial java/lang/Object.<init>:()V

- обращение к полям

  getfield, setfield, getstaticfield, setstaticfield

- работа с массивами

  *aload, *astore

Method Handles

Прямые указатели

- Указатели на методы
  - findStatic, findVirtual, findSpecial, findConstructor
- Указатели на поля
  - findGetter, findSetter, findStaticGetter, findStaticSetter
- Указатели на элементы массива
  - arrayElementGetter, arrayElementSetter

Method Handles

Манипулирование аргументами

- insert, drop, permute
- filter, fold, cast
- spread (unbox varargs)

Method Handles

Flow control

- guardWithTest: условный переход
  - комбинация 3х MH:
    - condition, true path, false path
- SwitchPoint: on/off branch
  - комбинация 2х MH
    - true and false paths
  - Once off, always off

Method Handles

Работа с исключениями

- catchException
  - body, exception type, handler
- throwException
  - throws Throwable in argument 0

Method Handles

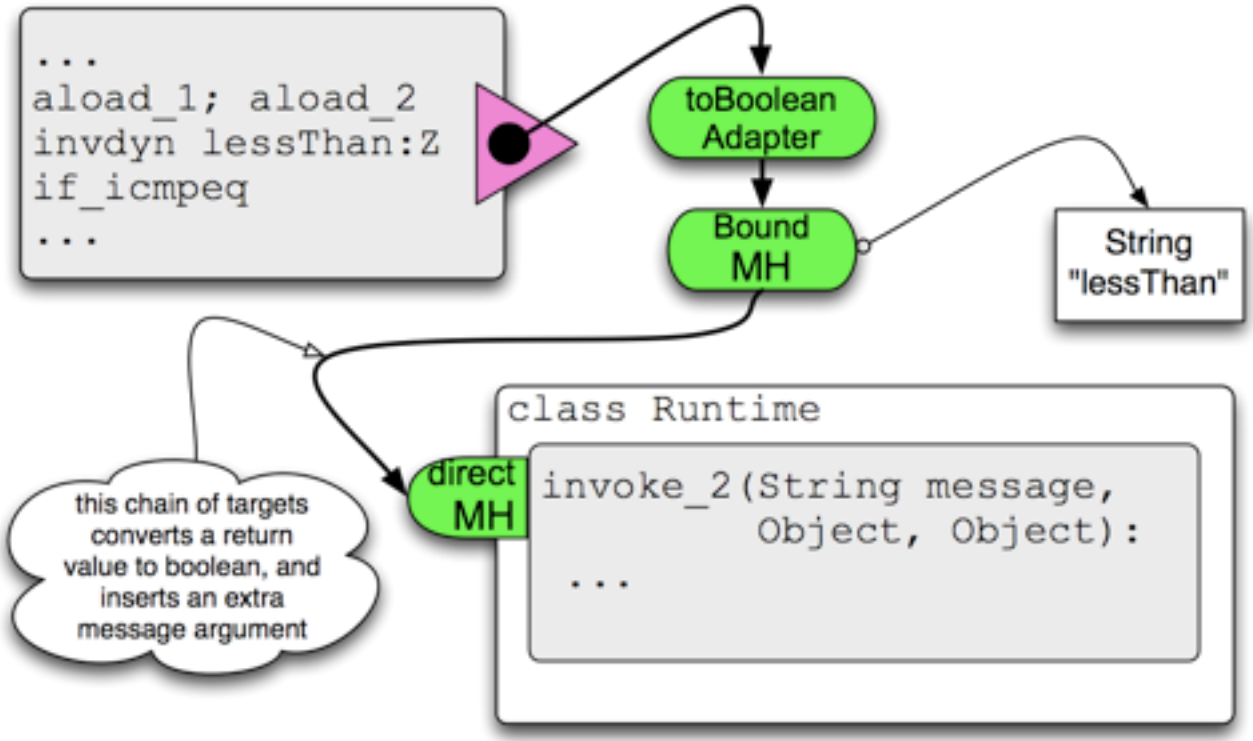
Жизненный цикл

- Создание **(direct MHs)**
  - reflective factory API: MethodHandles.Lookup
  - ldc of CONSTANT_MethodHandle
  - special factories: identity, invoker
- Трансформация/адаптация **(bound or adapter MHs)**
  - bindTo, insertArguments, guardWithTest, etc.
  - asType, filterArguments, etc.
- Вызов **(exact or inexact)**
- Линковка **(invokedynamic call site или некоторая константа)**

Method Handles

Типы

- **Direct Method Handle** указывает на Java метод.
  - может эмулировать существующие инструкции
- **Bound Method Handle** хранит значение параметра
  - связанный аргумент указывается при создании
  - используется при каждом вызове
  - любой MH может быть связан и это не видно вызывающему
- **Adapter Method Handle** преобразует значения «на лету»
  - как аргументы, так и возвращаемое значение
  - cast, box/unbox, collect/spread, filter, etc.
  - любой MH может быть адаптирован и это не видно вызывающему


...
aload_1; aload_2
invdyn lessThan:Z
if_icmpeq
...
toBoolean Adapter
Bound MH
String "lessThan"
class Runtime
direct MH
invoke_2(String message,
Object, Object):
...
this chain of targets converts a return value to boolean, and inserts an extra message argument

Method Handles

Жизненный цикл: Создание

- При создании method handle’а осуществляется и связывание с конкретным методом
- Может быть медленным
  - делайте только один раз, если возможно
- Используйте CONSTANT_MethodHandle
  - если генерируете байткод

Method Handles

Жизненный цикл: Вызов

- Осуществляется через методы, полиморфные относительно сигнатуры
- Сигнатура метода определяется в месте вызова (call site)
- Статически (в момент компиляции)
- Например, MethodHandle.invoke()
  - (int)mh.invoke(1, 1L, new Object()) =>(int, long, Object)int
- Сигнатура должна точно соответствовать типу MethodHandle’a
  - или нужно использовать invokeWithArgument
- Помечены @SignaturePolymorphic в коде

Method Handles

Жизненный цикл: Вызов

- «Точный» вызов (MH.invokeExact) прост
  - условный переход
  - немного дороже чем обычный виртуальный вызов.
- «Неточный» вызов (MH.invoke) может быть сложен
  - если типы не совпадают
  - иначе как invokeExact
- Вызов через invokedynamic инструкцию быстр
  - всегда «точен» (call site всегда строго типизирован)
  - код метода обычно инлайнится

Method Handles

Жизненный цикл: Связывание

- Инструкция invokedynamic
  - call site с оптимистично предсказанным вызовом метода
  - предсказанный method handle предполагается константой
- “static final” MH переменные рассматриваются как константы
- Константные method handle’ы могут быть заинлайнены
  - может позволить избавиться от адаптеров и связанных аргументов
  - может продолжаться до прямых указателей на Java методы

Method Handles

Пакет java.lang.invoke

- MethodHandle
  - прямой указатель + различные адаптации
- MethodType
  - сигнатура метода
- MethodHandles.Lookup
  - конструирование method handle’ов
- MethodHandles
  - набор вспомогательный функций для получения и адаптации method handle’ов

Method Handles

Примеры использования

```
MethodHandles.Lookup LOOKUP = MethodHandles.lookup();
```

Method Handles

Примеры использования

```
MethodHandles.Lookup LOOKUP = MethodHandles.lookup();

MethodHandle HASHCODE =
    LOOKUP. findStatic(
            System.class,
            "identityHashCode",
            MethodType.methodType(int.class, Object.class));
```

Method Handles

Примеры использования

```
MethodHandles.Lookup LOOKUP = MethodHandles.lookup();

MethodHandle HASHCODE =
    LOOKUP. findStatic(
                System.class,
                "identityHashCode",
                MethodType.methodType(int.class, Object.class));

assertEquals(
               System.identityHashCode("xy"),
               (int) HASHCODE.invokeExact("xy"));
```

Method Handles

Примеры использования

```
MethodHandles.Lookup LOOKUP = MethodHandles.lookup();

MethodHandle CONCAT =
    LOOKUP. findVirtual(
            String.class,
            "concat",
            MethodType.methodType(String.class, String.class));
```

Method Handles

Примеры использования

```
MethodHandles.Lookup LOOKUP = MethodHandles.lookup();

MethodHandle CONCAT =
    LOOKUP. findVirtual(
            String.class,
            "concat",
            MethodType.methodType(String.class, String.class));

assertEquals("xy", (String) CONCAT.invokeExact("x", "y"));
```

Method Handles

Примеры использования

```
MethodHandles.Lookup LOOKUP = MethodHandles.lookup();

MethodHandle CONCAT =
    LOOKUP. findVirtual(
            String.class,
            "concat",
            MethodType.methodType(String.class, String.class));

assertEquals("xy", (String) CONCAT.invokeExact("x", "y"));

MethodHandle CONCAT_x = CONCAT.bindTo(”x");
assertEquals(”xy", CONCAT_x.invoke(”y"));
```

Сравнение с Reflection API

- Быстрее чем Reflection API
  - проверки прав доступа при каждом вызове не нужны
    - осуществляются при **создании** MH
  - цепочки MH хорошо инлайнятся
- Удобнее чем Reflection API
  - комбинаторная библиотека для трансформации MH
  - точная типизация(по желанию)

Использование JSR292

Крупные проекты, активно использующие JSR292

- JRuby
- Project Lambda (Java 8)
- Nashorn (Java 8)

Project Lambda (Java 8)

"… We can achieve both of these goals [maximizing flexibility & providing stability] by using the invokedynamic feature from JSR 292 to separate the binary representation of lambda creation in the bytecode from the mechanics of evaluating the lambda expression at runtime. The use of invokedynamic lets us defer the selection of a translation strategy until run time. "

Brian Goetz

Java Language Architect,
Oracle, April, 2012

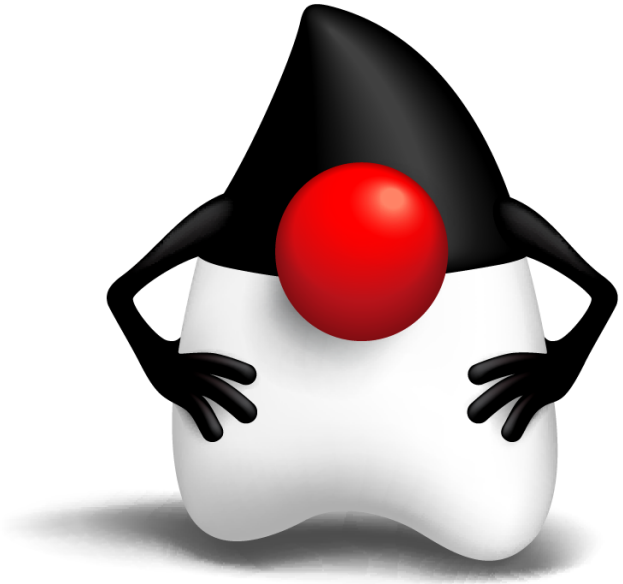
Итоги

- Роскошь? Далеко не всегда!
- Быстрее чем Reflection API
- Удобнее чем Reflection API

Использование

- invokedynamic доступна только на уровне байткода
  - через библиотеку ASM, например
- … но method handle’ы можно использовать в обычном Java коде!

Вопросы?



vladimir.x.ivanov@oracle.com
@iwanowww

MAKE THE FUTURE JAVA

ORACLE