Java™

ORACLE

A Renaissance VM:
One Platform, Many Languages

John R. Rose, Da Vinci Machine Project Lead

Announcing

Java 7 support for dynamic languages:

Invokedynamic

Overview...

- Why dynamic languages?
- The invokedynamic instruction
- Method handles
- User experience

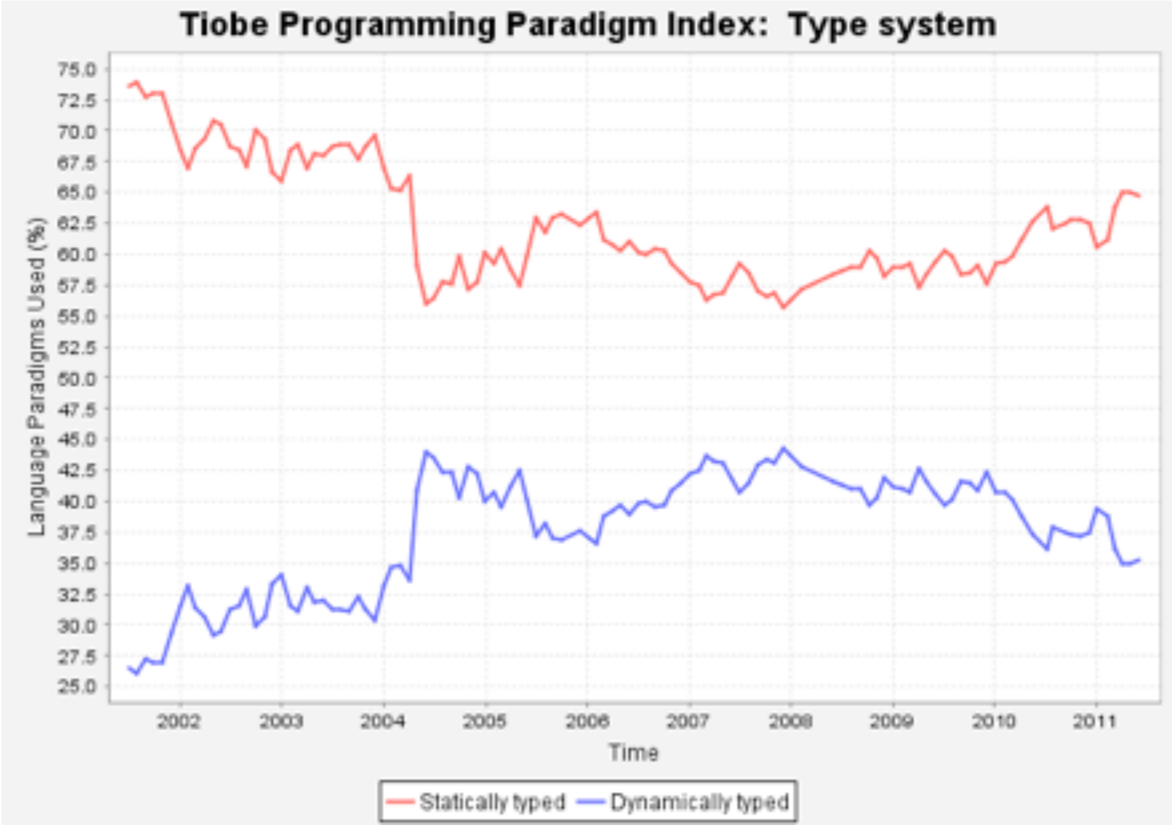
Why dynamic languages?

- Fast turnaround time for simple programs
  - no compile step required
  - direct interpretation possible
  - loose binding to the environment
- Data-driven programming
  - program shape can change along with data shape
  - radically open-ended code (plugins, aspects, closures)

Key dynamic languages on the JVM

- JavaScript (Rhino)
- Ruby (JRuby)
- Python (Jython)
- Lisp (Clojure, Kawa, ABCL, etc.)
- Groovy
- Smalltalk
- ...and many, many more

Dynamic languages are here to stay



Source: http://tiobe.com

“Businesses like Twitter, LinkedIn, and RedHat are increasing attention to Ruby because of its fast turnaround times... Implementations like JRuby have started to solve performance problems of the past.”

Charles Nutter

JRuby Lead, Engine Yard

Overview...

- Why dynamic languages?
- The invokedynamic instruction
- Method handles
- User experience

What a JVM can do...

- compiler tactics
  - delayed compilation
  - Tiered compilation
  - on-stack replacement
  - delayed reoptimization
  - program dependence graph representation
  - static single assignment representation
- proof-based techniques
  - exact type inference
  - memory value inference
  - memory value tracking
  - constant folding
  - reassociation
  - operator strength reduction
  - null check elimination
  - type test strength reduction
  - type test elimination
  - algebraic simplification
  - common subexpression elimination
  - integer range typing
- flow-sensitive rewrites
  - conditional constant propagation
  - dominating test detection
  - flow-carried type narrowing
  - dead code elimination
- language-specific techniques
  - class hierarchy analysis
  - devirtualization
  - symbolic constant propagation
  - autobox elimination
  - escape analysis
  - lock elision
  - lock fusion
  - de-reflection
- speculative (profile-based) techniques
  - optimistic nullness assertions
  - optimistic type assertions
  - optimistic type strengthening
  - optimistic array length strengthening
  - untaken branch pruning
  - optimistic N-morphic inlining
  - branch frequency prediction
  - call frequency prediction
- memory and placement transformation
  - expression hoisting
  - expression sinking
  - redundant store elimination
  - adjacent store fusion
  - card-mark elimination
  - merge-point splitting
- loop transformations
  - loop unrolling
  - loop peeling
  - safepoint elimination
  - iteration range splitting
  - range check elimination
  - loop vectorization
- global code shaping
  - inlining (graph integration)
  - global code motion
  - heat-based code layout
  - switch balancing
  - throw inlining
- control flow graph transformation
  - local code scheduling
  - local code bundling
  - delay slot filling
  - graph-coloring register allocation
  - linear scan register allocation
  - live range splitting
  - copy coalescing
  - constant splitting
  - copy removal
  - address mode matching
  - instruction peepholing
  - DFA-based code generator

...And what slows down a JVM

- Non-Java languages require special call sites.
  - Example: Smalltalk message sending (no static types).
  - Example: JavaScript or Ruby method call (different lookup rules).
- In the past, special calls required simulation overheads
  - ...such as reflection and/or extra levels of lookup and indirection
  - ...which have inhibited JIT optimizations.
- Result: Pain for non-Java developers.
- Enter Java 7.

Key Features

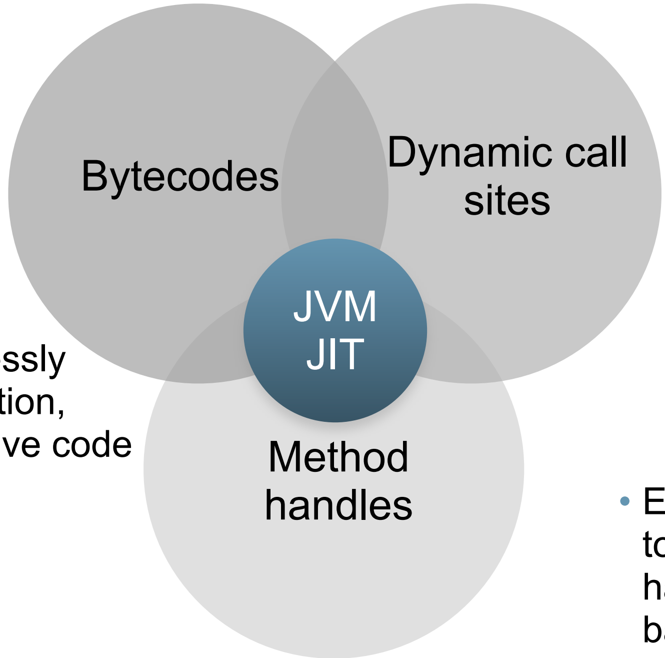
- New bytecode instruction: *invokedynamic*.
  - Linked reflectively, under user control.
  - User-visible object: java.lang.invoke.CallSite
  - Dynamic call sites can be linked and relinked, dynamically.

- New unit of behavior: *method handle*
  - The content of a dynamic call site is a method handle.
  - Method handles are function pointers for the JVM.

  - (Or if you like, each MH implements a single-method interface.)

Dynamic program composition

- Bytecodes are created by Java compilers or dynamic runtimes.
- A dynamic call site is created for each invokedynamic bytecode.



- The JVM seamlessly integrates execution, optimizing to native code as necessary.
- Each call site is bound to one or more method handles, which point back to bytecoded methods.

Passing the burden to the JVM

- Non-Java languages require special call sites.
- In the past, special calls required simulation overheads

- Now, invokedynamic call sites are fully user-configurable
  - ...and are fully optimizable by the JIT.
- Result: Much simpler code for language implementors
  - ...and new leverage for the JIT.

What's in a method call? *(before invokedynamic)*

	Source code	**Bytecode**	**Linking**	**Executing**
Naming	Identifiers	Utf8 constants	JVM "dictionary"	
Selecting	Scopes	Class names	Loaded classes	V-table lookup
Adapting	Argument conversion		C2I / I2C adapters	Receiver narrowing
Calling				*Jump with arguments*

What's in a method call? *(using invokedynamic)*

	Source code	Bytecode	Linking	Executing
Naming	∞	∞	∞	∞
Selecting	∞	Bootstrap methods	Bootstrap method call	∞
Adapting	∞		Method handles	∞
Calling				*Jump with arguments*

“Invokedynamic is the most important addition to Java in years. It will change the face of the platform.”

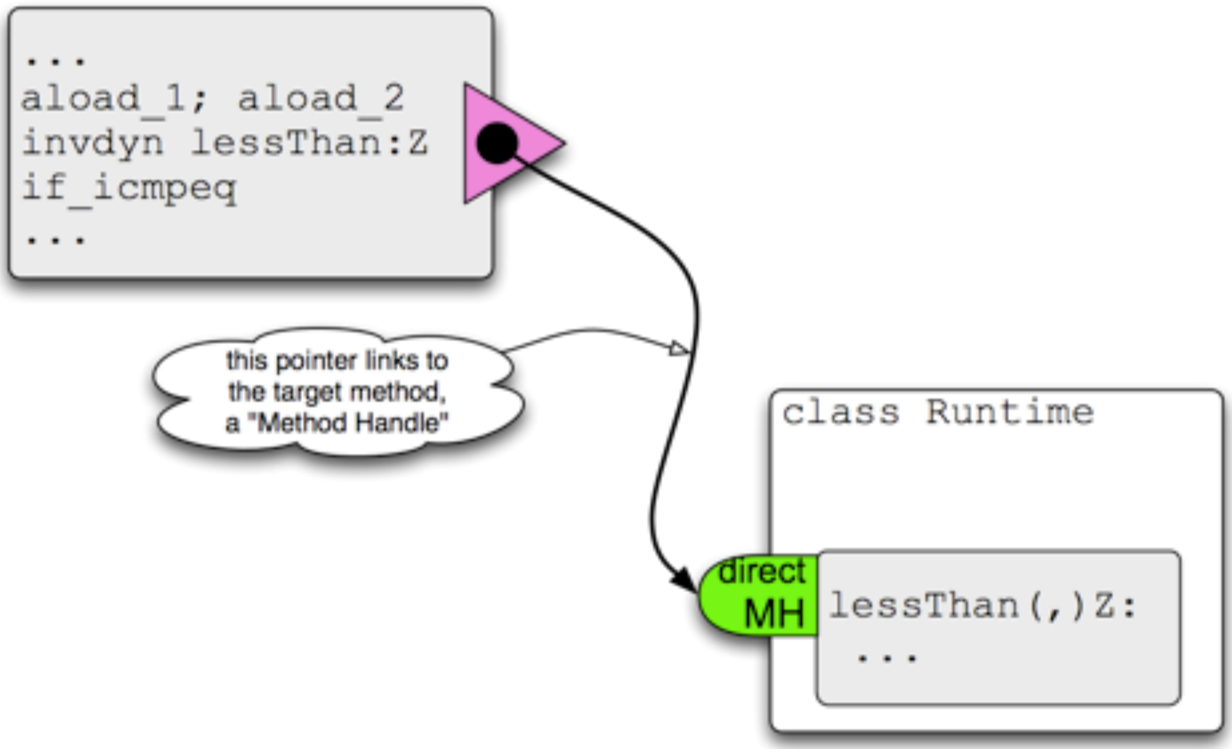
Charles Nutter

JRuby Lead, Engine Yard

Overview...

- Why dynamic languages?
- The invokedynamic instruction
- Method handles
- User experience

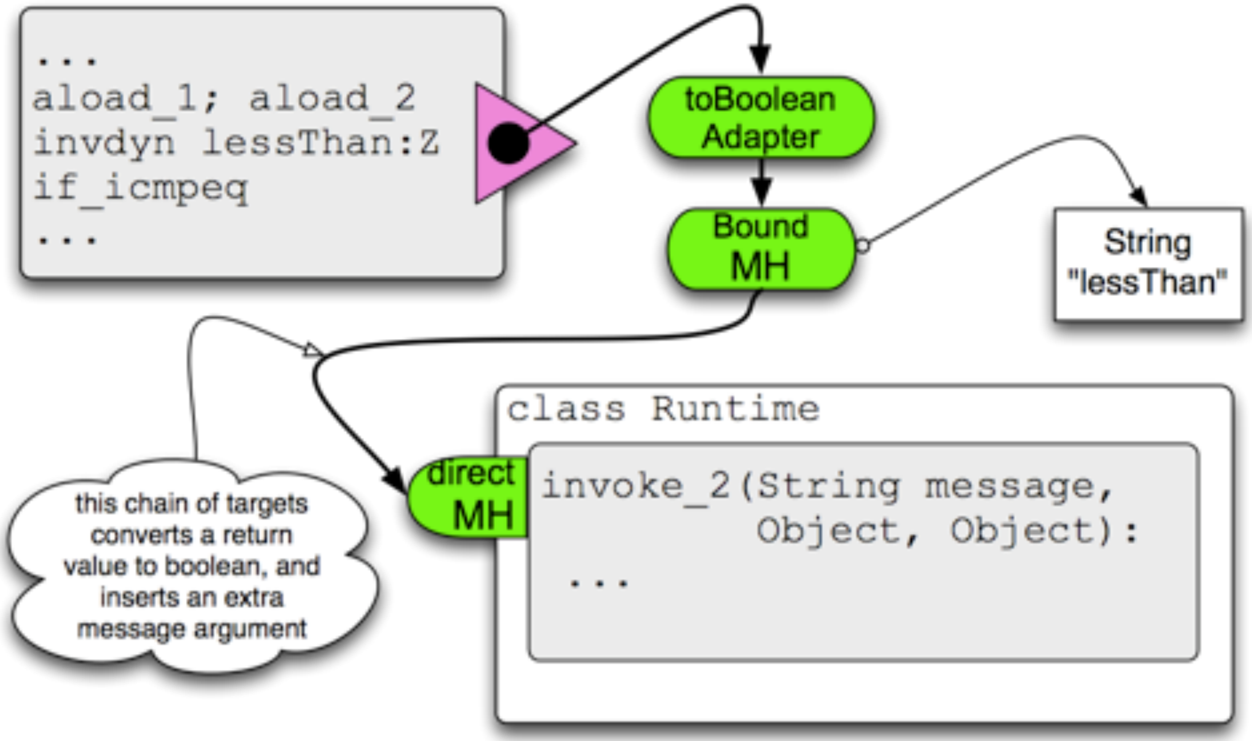
Invokedynamic “plumbing”, take 1



More details about method handles

- A *direct method handle* points to a Java method.
  - A DMH can emulate any of the pre-existing invoke instructions.
- A *bound method handle* includes an saved argument.
  - The bound argument is specified on creation, and is used on call.
  - The bound argument is inserted into the argument list.
  - Any MH can be be bound, and the binding is invisible to callers.
- An *adapter method handle* adjusts values on the fly.
  - Both argument and return values can be adjusted.
  - Adaptations include cast, box/unbox, collect/spread, filter, etc.
  - Any MH can be adapted. Adaptation is invisible to callers.

Invokedynamic “plumbing”, take 2



Coding directly with method handles

```
import static java.lang.invoke.MethodHandles.*;
import static java.lang.invoke.MethodType.*;
...
MethodHandles.Lookup LOOKUP = lookup();

MethodHandle HASHCODE = LOOKUP
  .findStatic(System.class,
    "identityHashCode", methodType(int.class, Object.class));
{assertEquals("xy".hashCode(), (int) HASHCODE.invoke("xy"));}

MethodHandle CONCAT = LOOKUP
  .findVirtual(String.class,
    "concat", methodType(String.class, String.class));
{assertEquals("xy", (String) CONCAT.invokeExact("x", "y"));}

MethodHandle CONCAT_FU = CONCAT.bindTo("fu");
{assertEquals("futbol", CONCAT_FU.invoke("tbol"));}
```

Overview...

- Why dynamic languages?
- The invokedynamic instruction
- Method handles
- User experience

“We were able to implement all of the Smalltalk constructs... using invokedynamic to execute Smalltalk code on the JVM. ...The ease of putting a true dynamic language on the JVM was a wonder in itself.”

Mark Roos

Roos Instruments, Inc.

What shall we build today?

- Smaller, simpler script engines on the JVM.
- New options for high-performance programming.
  - Multiple programming paradigms with full optimization.
  - Function pointers, self-adjusting code.
- The only limit is our community's imagination.

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Over the next hill...

- Project Lambda: mastering the multi-core
  – Direct support for closures via invokedynamic.
  – Better optimization of parallel, data-intensive programs.

And the next...

- Da Vinci Machine Project: an open source incubator for JVM futures
  – Yearly event: JVM Language Summit
  – http://openjdk.java.net/projects/mlvm/jvmlangsummit/

“Invokedynamic makes it possible for every static-typed operation on JVM to be dynamic... my mind boggles at the possibilities.”

Charles Nutter

JRuby Lead, Engine Yard

Java™