JVM Bytecode for Dummies

(and for the rest of you, as well)

Intro

- Charles Oliver Nutter
  - “JRuby Guy”
  - Sun Microsystems 2006-2009
  - Engine Yard 2009-
- Primarily responsible for compiler, perf
  - Lots of bytecode generation

Two Parts

- JVM Bytecode
  - Inspection
  - Generation
  - How it works
- JVM JIT
  - How it works
  - Monitoring
  - Assembly (don’t be scared!)

Two Parts

- JVM Bytecode
  - Inspection
  - Generation
  - How it works
- JVM JIT
  - How it works
  - Monitoring
  - Assembly (don't be scared!)

Today

Session 25141
Hilton Yosemite ABC
Wednesday 10AM

Bytecode Definition

- "... instruction sets designed for efficient execution by a software interpreter ..."
- "... suitable for further compilation into machine code.

Byte Code

- One-byte instructions
- 256 possible “opcodes”
- 200 in use on current JVMs
  - Room for more :-)
- Little variation since Java 1.0

Microsoft's CLR

- Stack-based, but not interpreted
- Two-byte "Wordcodes"
- Similar operations to JVM

Why Learn It

- Know your platform
  - Full understanding from top to bottom
- Bytecode generation is fun and easy
  - Build your own language?
- May need to read bytecode someday
  - Many libraries generate bytecode

Hello World

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

javap

- Java class file disassembler
- Basic operation shows class structure
  - Methods, superclasses, interface, etc
- -c flag includes bytecode
- -public, -private, -protected
- -verbose for stack size, locals, args

javap

```
~/projects/bytecode_for_dummies ➔ javap HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
    public HelloWorld();
    public static void main(java.lang.String[]);
}
```

javap -c

```
~/projects/bytecode_for_dummies ➔ javap -c HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0:  aload_0
   1:  invokespecial #1; //Method java/lang/Object."<init>":()V
   4:  return

public static void main(java.lang.String[]);
  Code:
   0:  getstatic  #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:  ldc #3; //String Hello, world
   5:  invokevirtual #4; //Method java/io/PrintStream.println:
                                                        (Ljava/lang/String;)V
   8:  return

}
```

javap -verbose

```
~/projects/bytecode_for_dummies ➔ javap -c -verbose HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object
  SourceFile: "HelloWorld.java"
  minor version: 0
  major version: 50
  Constant pool:
const #1 = Method   #6.#15; //  java/lang/Object."<init>":()V
const #2 = Field#16.#17;//  java/lang/System.out:Ljava/io/PrintStream;
const #3 = String   #18;//  Hello, world
const #4 = Method   #19.#20;//  java/io/PrintStream.println:(Ljava/lang/String;)V
const #5 = class#21;//  HelloWorld
...

{
```

javap -verbose

```
...
public HelloWorld();
  Code:
   Stack=1, Locals=1, Args_size=1
   0: aload_0
   1: invokespecial  #1; //Method java/lang/Object."<init>":()V
   4: return
  LineNumberTable:
   line 1: 0
```

javap -verbose

```
public static void main(java.lang.String[]);
  Code:
   Stack=2, Locals=1, Args_size=1
   0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:  ldc #3; //String Hello, world
   5:  invokevirtual #4; //Method java/io/PrintStream.println:
                                                    (Ljava/lang/String;)V
   8:  return
  LineNumberTable:
   line 3: 0
   line 4: 8


}
```

TraceClassVisitor

```
$ java -cp <ASM stuff> org.objectweb.asm.util.TraceClassVisitor HelloWorld.class
// class version 50.0 (50)
// access flags 33
public class HelloWorld {


  // access flags 1
  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 9
  public static main([Ljava/lang/String;)V
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    LDC "Hello, world"
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
    RETURN
    MAXSTACK = 2
    MAXLOCALS = 1
}
```

ASMifierClassVisitor

```
$ java -cp <ASM stuff> org.objectweb.asm.util.ASMifierClassVisitor HelloWorld.class
import java.util.*;
import org.objectweb.asm.*;
import org.objectweb.asm.attrs.*;
public class HelloWorldDump implements Opcodes {

public static byte[] dump () throws Exception {

ClassWriter cw = new ClassWriter(0);
FieldVisitor fv;
MethodVisitor mv;
AnnotationVisitor av0;

cw.visit(V1_6, ACC_PUBLIC + ACC_SUPER, "HelloWorld", null, "java/lang/Object", null);
...
```

ASMifierClassVisitor

```
...
{
mv = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null);
mv.visitCode();
mv.visitVarInsn(ALOAD, 0);
mv.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V");
mv.visitInsn(RETURN);
mv.visitMaxs(1, 1);
mv.visitEnd();
}
{
mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "main", "([Ljava/lang/String;)V", null, null);
mv.visitCode();
mv.visitFieldInsn(GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;");
mv.visitLdcInsn("Hello, world");
mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", "(Ljava/lang/
String;)V");
mv.visitInsn(RETURN);
mv.visitMaxs(2, 1);
mv.visitEnd();
}
cw.visitEnd();

return cw.toByteArray();
}
}
```

Thank you!

Thank you!

(Just Kidding)

Let’s try something a little easier...

BiteScript

- (J)Ruby DSL for emitting JVM bytecode
  - Internal DSL
  - Primitive “macro” support
  - Reads like javap -c (but nicer)
- http://github.com/headius/bitescript

Installation

- Download JRuby from http://jruby.org
- Unpack, optionally add bin/ to PATH
  - Ahead of PATH if you have Ruby already
- [bin/]jruby -S gem install bitescript
- `bite myfile.bs` to run myfile.bs file
- `bitec myfile.bs` to compile myfile.bs file

BiteScript Users

- Mirah
  - Ruby-like language for writing Java code
  - BiteScript for JVM bytecode backend
- BrainF*ck implementation
- Other miscellaneous bytecode experiments

JiteScript

- Java API that mimics BiteScript
  - Using a few cute tricks ;-)
- Pretty close to javap output
- Typical Java library installation
- http://github.com/qmx/jitescript

JiteScript Users

- dyn.js
  - invokedynamic-based JavaScript impl
- ???

javap -c

```
~/projects/bytecode_for_dummies ➔ javap -c HelloWorld
Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object{
public HelloWorld();
  Code:
   0:  aload_0
   1:  invokespecial #1; //Method java/lang/Object."<init>":()V
   4:  return

public static void main(java.lang.String[]);
  Code:
   0:  getstatic  #2; //Field java/lang/System.out:Ljava/io/PrintStream;
   3:  ldc #3; //String Hello, world
   5:  invokevirtual #4; //Method java/io/PrintStream.println:
                                                        (Ljava/lang/String;)V
   8:  return

}
```

BiteScript

```
main do
  getstatic java.lang.System, "out",
            java.io.PrintStream
  ldc "Hello, world!"
  invokevirtual java.io.PrintStream, "println",
      [java.lang.Void::TYPE, java.lang.Object]
  returnvoid
end
```

BiteScript

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println", [void, object]
  returnvoid
end
```

JRuby's "import" for Java classes

Shortcuts for void, int, string, object, etc

BiteScript

```
main do
  ldc "Hello, world!"
  aprintln
  returnvoid
end
```

A BiteScript “macro”

BiteScript

```
macro :aprintln do
  getstatic System, "out", PrintStream
  swap
  invokevirtual PrintStream, "println",
                [void, object]
end
```

The Basics

- Stack machine
- Basic operations
- Flow control
- Class structures
- Exception handling

Stack Machine

- The “operand stack” holds operands
- Operations push and/or pop stack values
  - Exceptions: nop, wide, goto, jsr/ret
- Stack must be consistent
  - Largest part of bytecode verifier
- Stack is explicitly sized per method

The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                                [void, object]
  returnvoid
end
```

Depth	Value
0	
1	
2	
3	
4	

The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                                [void, object]
  returnvoid
end
```

Depth	Value
0	**out (a PS)**
1	
2	
3	
4	

The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                                [void, object]
  returnvoid
end
```

Depth	Value
0	**“Hello, world!”**
1	out (a PS)
2	
3	
4	

The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                                [void, object]
  returnvoid
end
```

Depth	Value
0	
1	
2	
3	
4	

The JVM Stack

```
import java.lang.System
import java.io.PrintStream

main do
  getstatic System, "out", PrintStream
  ldc "Hello, world!"
  invokevirtual PrintStream, "println",
                                [void, object]
  returnvoid
end
```

Depth	Value
0	
1	
2	
3	
4	

Basic Operations

- Stack manipulation
- Local variables
- Math
- Boolean

Stack Operations

0x00	nop	Do nothing.
0x57	pop	Discard top value from stack
0x58	pop2	Discard top two values
0x59	dup	Duplicate and push top value again
0x5A	dup_x1	Dup and push top value below second value
0x5B	dup_x2	Dup and push top value below third value
0x5C	dup2	Dup top two values and push
0x5D	dup2_x1	...below second value
0x5E	dup2_x2	...below third value
0x5F	swap	Swap top two values

Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

Depth	Value
0	value_0
1	value_1
2	
3	
4	

Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

Depth	Value
0	**value_0**
1	value_0
2	value_1
3	
4	

Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

Depth	Value
0	value_0
1	value_1
2	
3	
4	

Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

Depth	Value
0	**value_1**
1	**value_0**
2	
3	
4	

Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

Depth	Value
0	value_1
1	value_0
2	**value_1**
3	
4	

Stack Juggling

dup
pop
swap
dup_x1
dup2_x2

Depth	Value
0	value_1
1	value_0
2	value_1
3	**value_1**
4	**value_0**

Typed Opcodes

`<type><operation>`

b	byte
s	short
c	char
i	int
l	long
f	float
d	double
a	reference

Constant values
Local vars (load, store)
Array operations (aload, astore)
Math ops (add, sub, mul, div)
Boolean and bitwise
Comparisons
Conversions

Where's boolean?

- Boolean is generally int 0 or 1
- Boolean operations push int 0 or 1
- Boolean branches expect 0 or nonzero
- To set a boolean...use int 0 or 1

Constant Values

0x01	aconst_null	Push null on stack
0x02-0x08	iload_[m1-5]	Push integer [-1 to 5] on stack
0x09-0x0A	lconst_[0,1]	Push long [0 or 1] on stack
0x0B-0x0D	fconst_[0,1,2]	Push float [0.0, 1.0, 2.0] on stack
0x0E-0x0F	dconst_[0,1]	Push double [0.0, 1.0] on stack
0x10	bipush	Push byte value to stack as integer
0x11	sipush	Push short value to stack as integer
0x12	ldc	Push 32-bit constant to stack (int, float, string)
0x14	ldc2_w	Push 64-bit constant to stack (long, double)

Why So Many?

- Reducing bytecode size
  - Special iconst_0 and friends take no args
  - bipush, sipush: only 8, 16 bits arguments
- Pre-optimizing JVM
  - Specialized instructions can be optimized
  - Doesn’t matter at all now

Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

Depth	Value
0	
1	
2	
3	
4	
5	

Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

Depth	Value
0	**“hello”**
1	
2	
3	
4	
5	

Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

Depth	Value
0	**1.0d**
1	
2	“hello”
3	
4	
5	

Woah, Two Slots?

- JVM stack slots (and local vars) are 32-bit
- 64-bit values take up two slots
- “wide” before or “w” suffix
- 64-bit field updates not atomic!
  - Mind those concurrent longs/doubles!

Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

Depth	Value
0	**null**
1	1.0d
2	
3	“hello”
4	
5	

Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

Depth	Value
0	**4**
1	null
2	1.0d
3	
4	“hello”
5	

Constant Values

```
ldc "hello"
dconst_1
aconst_null
bipush 4
ldc_float 2.0
```

Depth	Value
0	**2.0f**
1	4
2	null
3	1.0
4	
5	“hello”

Local Variable Table

- Local variables numbered from 0
  - Instance methods have “this” at 0
- Separate table maps numbers to names
- Explicitly sized in method definition

Local Variables

0x15	iload	Load integer from local variable onto stack
0x16	lload	...long...
0x17	fload	...float...
0x18	dload	...double...
0x19	aload	...reference...
0x1A-0x2D	Packed loads	iload_0, aload_3, etc
0x36	istore	Store integer from stack into local variable
0x37	lstore	...long...
0x38	fstore	...float...
0x39	dstore	...double...
0x3A	astore	...reference...
0x3B-0x4E	Packed stores	fstore_2, dstore_0, etc
0x84	iinc	Add given amount to int local variable

Local Variables

Var	Value
0	
1	
2	
3	
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	
1	
2	
3	
4	

Local Variables

Var	Value
0	
1	
2	
3	
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	**“hello”**
1	
2	
3	
4	

Local Variables

Var	Value
0	
1	
2	
3	
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	**4**
1	“hello”
2	
3	
4	

Local Variables

Var	Value
0	
1	
2	
3	**4**
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	“hello”
1	
2	
3	
4	

Local Variables

Var	Value
0	
1	
2	
3	4
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	**0.0**
1	
2	“hello”
3	
4	

Local Variables

Var	Value
0	
1	**0.0**
2	
3	4
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	“hello”
1	
2	
3	
4	

Local Variables

Var	Value
0	**“hello”**
1	0.0
2	
3	4
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	
1	
2	
3	
4	

Local Variables

Var	Value
0	“hello”
1	0.0
2	
3	4
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	**“hello”**
1	
2	
3	
4	

Local Variables

Var	Value
0	“hello”
1	0.0
2	
3	**9**
4	

```
ldc "hello"
bipush 4
istore 3
dconst_0
dstore 1
astore 0
aload 0
iinc 3, 5
```

Depth	Value
0	“hello”
1	
2	
3	
4	

Arrays

0x2E-0x35	[i,l,f,d,a,b,c,d]aload	Load [int, long, ...] from array (on stack) to stack
0x4F-0x56	[i,l,f,d,a,b,c,d]astore	Store [int, long, ...] from stack to array (on stack)
0xBC	newarray	Construct new primitive array
0xBD	anewarray	Construct new reference array
0xBE	arraylength	Get array length
0xC5	multianewarray	Create multi-dimensional array

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	
1	
2	
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	**2**
1	
2	
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	**int[2] {0,0}**
1	
2	
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	**int[2] {0,0}**
1	int[2] {0,0}
2	
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	**0**
1	int[2] {0,0}
2	int[2] {0,0}
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	-1
1	0
2	int[2] {0,0}
3	int[2] {0,0}
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	int[2] {-1, 0}
1	
2	
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	**0**
1	int[2] {-1, 0}
2	
3	
4	
5	

Arrays

```
iconst_2
newarray int
dup
iconst_0
iconst_m1
iastore
iconst_0
iaload
```

Depth	Value
0	-1
1	
2	
3	
4	
5	

Math Operations

	add +	subtract -	multiply *	divide /	remainder %	negate -()
int	iadd	isub	imul	idiv	irem	ineg
long	ladd	lsub	lmul	ldiv	lrem	lneg
float	fadd	fsub	fmul	fdiv	frem	fneg
double	dadd	dsub	dmul	ddiv	drem	dneg

Boolean and Bitwise

	shift left	shift right	unsigned shift right	and	or	xor
int	ishl	ishr	iushr	iand	ior	ixor

Conversions

To:

From:

	int	long	float	double	byte	char	short
int	-	i2l	i2f	i2d	i2b	i2c	i2s
long	l2i	-	l2f	l2d	-	-	-
float	f2i	f2l	-	f2d	-	-	-
double	d2i	d2l	d2f	-	-	-	-

Comparisons

0x94	lcmp	Compare two longs, push int -1, 0, 1
0x95	fcmpl	Compare two floats, push in -1, 0, 1 (-1 for NaN)
0x96	fcmpg	Compare two floats, push in -1, 0, 1 (1 for NaN)
0x97	dcmpl	Compare two doubles, push in -1, 0, 1 (-1 for NaN)
0x98	dcmpg	Compare two doubles, push in -1, 0, 1 (1 for NaN)

Flow Control

- Inspect stack and branch
  - Or just branch, via goto
- Labels mark branch targets
- Wide variety of tests

Flow Control

0x99	ifeq	If zero on stack, branch
0x9A	ifne	If nonzero on stack, branch
0x9B	iflt	If stack value is less than zero, branch
0x9C	ifge	If stack value is greater than or equal to zero, branch
0x9D	ifgt	If stack value is greater than zero, branch
0x9E	ifle	If stack value is less than or equal to zero, branch
0x9F	if_icmpeq	If two integers on stack are eq, branch
0xA0	if_icmpne	If two integers on stack are ne, branch
0xA1	if_icmplt	If two integers on stack are lt, branch
0xA2	if_icmpge	If two integers on stack are ge, branch
0xA3	if_icmpgt	If two integers on stack are gt, branch
0xA4	if_icmple	If two integers on stack are le, branch
0xA5	if_acmpeq	If two references on stack are the same, branch
0xA6	if_acmpne	If two references on stack are different, branch
0xA7	goto	GOTO!

Other Flow Control

0xA8	jsr	Jump to subroutine (deprecated)
0xA9	ret	Return from subroutine (deprecated)
0xAA	tableswitch	Branch using an indexed table of jump offsets
0xAB	lookupswitch	Branch using a lookup-based table of jump offsets
0xAC-0xB0	[i,l,f,d,a]return	Return (int, long, float, double, reference) value
0xB1	return	Void return (exit method, return nothing)
0xC6	ifnull	If reference on stack is null
0xC7	ifnonnull	If reference on stack is not null

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	**String[] {“branch”}**
1	
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	**0**
1	String[]{“branch”}
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	**“branch”**
1	
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	**“branch”**
1	“branch”
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	I
1	
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
 ldc "Not equal!"
 aprintln
 goto :end
 label :branch
 ldc "Equal!"
 aprintln
 label :end
 returnvoid
```

Depth	Value
0	
1	
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
              [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	
1	
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	**“Equal!”**
1	
2	
3	
4	
5	

Flow Control

```
aload 0
ldc 0
aaload
ldc "branch"
invokevirtual string, "equals",
                [boolean, object]
ifne :branch
ldc "Not equal!"
aprintln
goto :end
label :branch
ldc "Equal!"
aprintln
label :end
returnvoid
```

Depth	Value
0	
1	
2	
3	
4	
5	

Classes and Types

- Signatures!!!
  - Probably the most painful part
  - ...but not a big deal if you understand

Using Classes

0xB2	getstatic	Fetch static field from class
0xB3	putstatic	Set static field in class
0xB4	getfield	Get instance field from object
0xB5	setfield	Set instance field in object
0xB6	invokevirtual	Invoke instance method on object
0xB7	invokespecial	Invoke constructor or “super” on object
0xB8	invokestatic	Invoke static method on class
0xB9	invokeinterface	Invoke interface method on object
0xBA	invokedynamic	Invoke method dynamically on object (Java 7)
0xBB	new	Construct new instance of object
0xC0	checkcast	Attempt to cast object to type
0xC1	instanceof	Push nonzero if object is instanceof specified type

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**an ArrayList (uninitialized)**
1	
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**an ArrayList (uninitialized)**
1	an ArrayList (uninitialized)
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

Depth	Value
0	an ArrayList
1	
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**a Collection**
1	
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**a Collection**
1	a Collection
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**“first element”**
1	a Collection
2	a Collection
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**1 (true)**
1	a Collection
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

Depth	Value
0	a Collection
1	
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

Depth	Value
0	**an ArrayList**
1	
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                  [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**0**
1	an ArrayList
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
                [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                  [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
                [object, int]
aprintln
returnvoid
```

Depth	Value
0	**“first element”**
1	
2	
3	
4	
5	

Using Classes

```
new ArrayList
dup
invokespecial ArrayList, '<init>',
              [void]
checkcast Collection
dup
ldc "first element"
invokeinterface Collection, 'add',
                [boolean, object]
pop
checkcast ArrayList
ldc 0
invokevirtual ArrayList, 'get',
              [object, int]
aprintln
returnvoid
```

Depth	Value
0	
1	
2	
3	
4	
5	

Invokedynamic

- New bytecode in Java 7
- Target method is wired up by user code
- Method handles are the wiring

Emitting Invokedynamic

- Signature is still required
  - But can be almost anything
- Method name is still required
  - But can be almost anything
- MethodHandle for bootstrapping
  - Bytecode-level function pointer, basically

java.lang.invoke

- MethodHandles
  - Function points
  - Adapters (arg juggling, catch, conditionals)
- CallSites
  - Place to bind your MH chain
- SwitchPoint
  - Zero-cost volatile boolean branch

```
import java.lang.invoke.MethodHandle
import java.lang.invoke.MethodType
import java.lang.invoke.CallSite
import java.lang.invoke.ConstantCallSite
import java.lang.invoke.MethodHandles::Lookup
JClass = java.lang.Class
```

Target Method

```
# The method we want to invoke, prints given string
public_static_method :print, [], void, string do
  aload 0
  aprintln
  returnvoid
end
```

Invokedynamic

```
# Our main method, which does one invokedynamic
main do
  # handle for our bootstrap, which binds invokedynamic to a CallSite
  bootstrap = mh_invokestatic this, 'bootstrap',
                              CallSite, Lookup, string, MethodType

  ldc 'Hello, invokedynamic!'
  invokedynamic 'print', [void, string], bootstrap
  returnvoid
end
```

Invokedynamic

```
# Our main method, which does one invokedynamic
main do
  # handle for our bootstrap, which binds invokedynamic to a CallSite
  bootstrap = mh_invokestatic this, 'bootstrap',
                              CallSite, Lookup, string, MethodType

  ldc 'Hello, invokedynamic!'
  invokedynamic 'print', [void, string], bootstrap
  returnvoid
end
```

Invokedynamic

```
# Our main method, which does one invokedynamic
main do
  # handle for our bootstrap, which binds invokedynamic to a CallSite
  bootstrap = mh_invokestatic this, 'bootstrap',
                              CallSite, Lookup, string, MethodType

  ldc 'Hello, invokedynamic!'
  invokedynamic 'print', [void, string], bootstrap
  returnvoid
end
```

Invokedynamic

```
# Our main method, which does one invokedynamic
main do
  # handle for our bootstrap, which binds invokedynamic to a CallSite
  bootstrap = mh_invokestatic this, 'bootstrap',
                              CallSite, Lookup, string, MethodType

  ldc 'Hello, invokedynamic!'
  invokedynamic 'print', [void, string], bootstrap
  returnvoid
end
```

Bootstrap

```
# The bootstrap method, which binds our dynamic call
public_static_method :bootstrap, [], CallSite,
                     Lookup, string, MethodType do
  # Constant since we bind just once directly
  new ConstantCallSite
  dup

  # Locate the method indicated by name + type on current class
  aload 0   # Lookup
  ldc this  # this class
  aload 1   # String
  aload 2   # MethodType
  invokevirtual Lookup, 'findStatic',
                [MethodHandle, JClass, string, MethodType]

  # finish constructing call site and return
  invokespecial ConstantCallSite, '<init>', [void, MethodHandle]
  areturn
end
```

Bootstrap

```
# The bootstrap method, which binds our dynamic call
public_static_method :bootstrap, [], CallSite,
                     Lookup, string, MethodType do
  # Constant since we bind just once directly
  new ConstantCallSite
  dup

  # Locate the method indicated by name + type on current class
  aload 0   # Lookup
  ldc this  # this class
  aload 1   # String
  aload 2   # MethodType
  invokevirtual Lookup, 'findStatic',
                [MethodHandle, JClass, string, MethodType]

  # finish constructing call site and return
  invokespecial ConstantCallSite, '<init>', [void, MethodHandle]
  areturn
end
```

Bootstrap

```
# The bootstrap method, which binds our dynamic call
public_static_method :bootstrap, [], CallSite,
                     Lookup, string, MethodType do
  # Constant since we bind just once directly
  new ConstantCallSite
  dup

  # Locate the method indicated by name + type on current class
  aload 0   # Lookup
  ldc this  # this class
  aload 1   # String
  aload 2   # MethodType
  invokevirtual Lookup, 'findStatic',
                [MethodHandle, JClass, string, MethodType]

  # finish constructing call site and return
  invokespecial ConstantCallSite, '<init>', [void, MethodHandle]
  areturn
end
```

Bootstrap

```
# The bootstrap method, which binds our dynamic call
public_static_method :bootstrap, [], CallSite,
                     Lookup, string, MethodType do
  # Constant since we bind just once directly
  new ConstantCallSite
  dup

  # Locate the method indicated by name + type on current class
  aload 0   # Lookup
  ldc this  # this class
  aload 1   # String
  aload 2   # MethodType
  invokevirtual Lookup, 'findStatic',
                [MethodHandle, JClass, string, MethodType]

  # finish constructing call site and return
  invokespecial ConstantCallSite, '<init>', [void, MethodHandle]
  areturn
end
```

Bootstrap

```
# The bootstrap method, which binds our dynamic call
public_static_method :bootstrap, [], CallSite,
                     Lookup, string, MethodType do
  # Constant since we bind just once directly
  new ConstantCallSite
  dup

  # Locate the method indicated by name + type on current class
  aload 0   # Lookup
  ldc this  # this class
  aload 1   # String
  aload 2   # MethodType
  invokevirtual Lookup, 'findStatic',
                [MethodHandle, JClass, string, MethodType]

  # finish constructing call site and return
  invokespecial ConstantCallSite, '<init>', [void, MethodHandle]
  areturn
end
```

Exceptions and Synchronization

-	trycatch	Table structure for a method indicating start/end of try/catch and logic to run on exception
0xC2	monitorenter	Enter synchronized block against object on stack
0xC3	monitorexit	Exit synchronized block (against same object)

More Examples

- A simple loop
- Fibonacci

A Simple Loop

```
main do
  aload 0
  push_int 0
  aaload
  label :top
  dup
  aprintln
  goto :top
  returnvoid
end
```

Fibonacci

```
public_static_method "fib", [], int, int do
  iload 0
  ldc 2
  if_icmpge :recurse
  iload 0
  ireturn
  label :recurse
  iload 0
  ldc 1
  isub
  invokestatic this, "fib", [int, int]
  iload 0
  ldc 2
  isub
  invokestatic this, "fib", [int, int]
  iadd
  ireturn
end
```

Fibonacci

```
main do
  load_times
  istore 1

  ldc "Raw bytecode fib(45) performance:"
  aprintln

  label :top
  iload 1
  ifeq :done
  iinc 1, -1

  start_timing 2
  ldc 45
  invokestatic this, "fib", [int, int]
  pop
  end_timing 2

  ldc "Time: "
  aprintln
  lprintln 2
  goto :top

  label :done
  returnvoid
end
```

Fibonacci

```
main do
  load_times
  istore 1

  ldc "Raw bytecode fib(45) performance:"
  aprintln

  label :top
  iload 1
  ifeq :done
  iinc 1, -1

  start_timing 2
  ldc 45
  invokestatic this, "fib", [int, int]
  pop
  end_timing 2

  ldc "Time: "
  aprintln
  lprintln 2
  goto :top

  label :done
  returnvoid
end
```

Macros

Fibonacci

```
macro :load_times do
  aload 0
  ldc 0
  aaload # number of times
  invokestatic JInteger, 'parseInt',
               [int, string]
end
```

Fibonacci

```
macro :start_timing do |i|
  load_time
  lstore i
end
```

Fibonacci

```
macro :load_time do
  invokestatic System, "currentTimeMillis", long
end
```

Fibonacci

```
macro :end_timing do |i|
  load_time
  lload i
  lsub
  lstore i
end
```

Fibonacci

```
macro :lprintln do |i|
  getstatic System, "out", PrintStream
  lload i
  invokevirtual PrintStream, "println",
                [void, long]
end
```

Real-world Cases

- Reflection-free invocation
  - JRuby, Groovy, other languages
- Bytecoded data objects
  - Hibernate, other data layers
  - java.lang.reflect.Proxy and others
- Language compilers

Tools

- BiteScript
  - So much fun
  - Ruby, so that's even better
- JiteScript
- ASM - defacto standard library

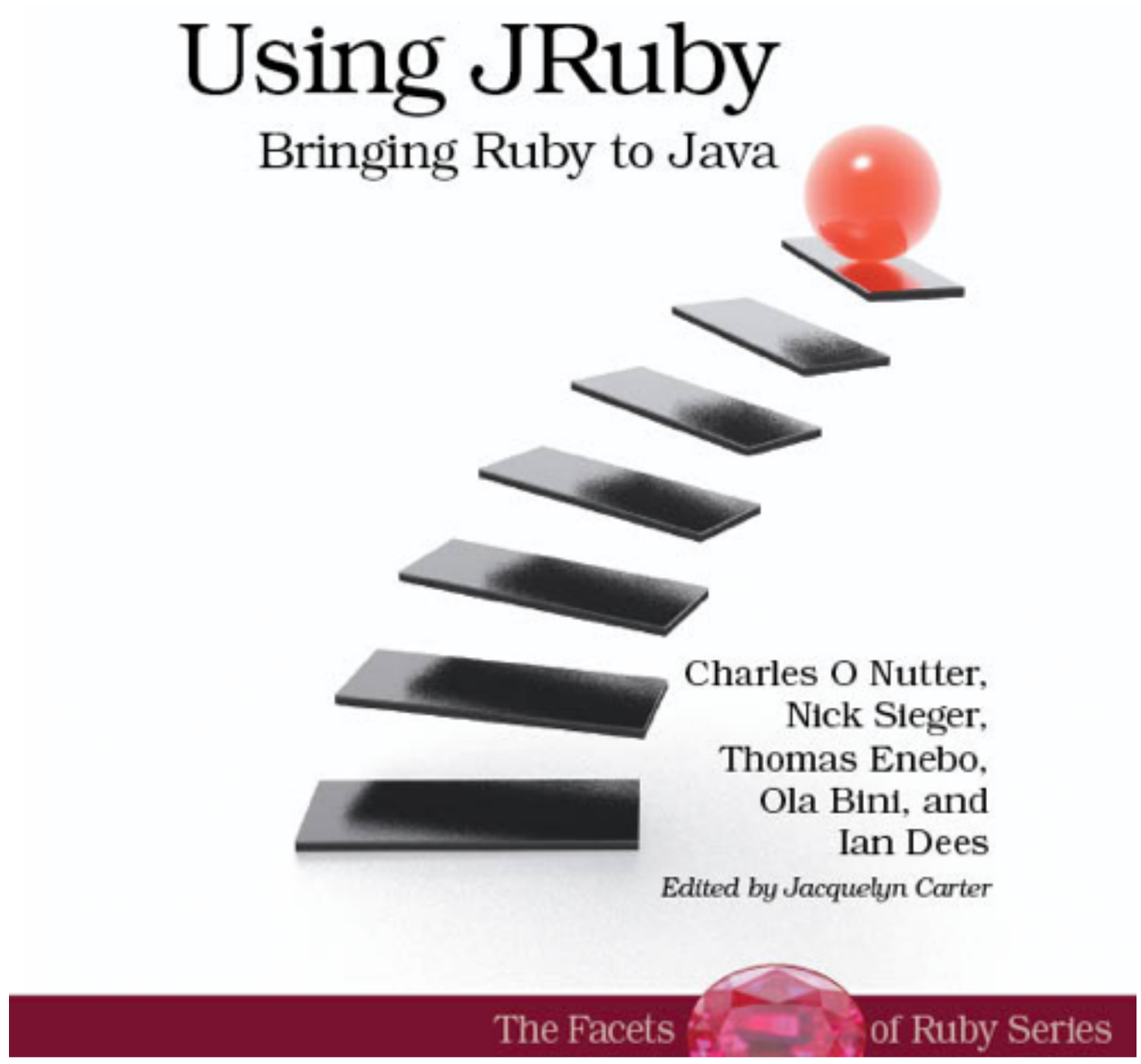
Part 2 Topics

- Tracking your bytecode through the JVM
- How the JVM optimizes running code
- Monitoring JVM JIT compilation
- Inspecting JVM inlining
- Dumping JVM JIT assembly output
- x86 assembler language for dummies!

Pluggity

- Many thanks to Engine Yard
  - JRuby and Java PaaS
  - JRuby support and services
  - sales@engineyard.com

Pluggity


The Pragmatic Programmers
Using JRuby
Bringing Ruby to Java
Charles O Nutter,
Nick Sieger,
Thomas Enebo,
Ola Bini, and
Ian Dees
Edited by Jacquelyn Carter
The Facets of Ruby Series

Pluggity

- Migrating to JRuby on Rails
  5:30 Mon, Parc 55 Market Street
- JVM JIT for Dummies ← Part 2
  10:00 Wed, Parc 55 Market street
- JRuby + Java + Cloud
  3:00 Wed, Parc 55 Embarcadero
- Real World JRuby
  4:30 Wed, Parc 55 Market Street

Pluggity

- JRuby meetup/party
  Engine Yard HQ
  500 Third Street, Suite 510
  Tuesday, 6:30PM
- Try JRuby on EY Cloud
  http://engineyard.com/tryjruby

Thank you!

- headius@headius.com, @headius
- http://blog.headius.com
- http://github.com/headius/bitescript
- “java virtual machine specification”
- “jvm opcodes”