

Java 7 bytecode:

slicing, dicing, and poking with a stick

Dawid WEISS

Carrot Search s.c., Poland

Talk outline

Syntactic sugar

What happens? At what level?

HotSpot improvements

GC, tiered jit compiler, invokedynamic

Experiments

Results of minor benchmarks comparing against 1.6

Syntactic sugar

This discussion based on javac/ HotSpot only.

Strings in switches

```
public static void main(String[] args) {  
    for (String s : args) {  
        switch (s) {  
            case "abc":  
                System.out.println("String 1");  
                break;  
            case "bde":  
                System.out.println("String 2");  
                break;  
        }  
    }  
}
```

Strings in switches

```
public static void main(String[] args) {  
    for (String s : args) {  
        switch (s) {  
            case "abc":  
                System.out.println("String 1");  
                break;  
            case "bde":  
                System.out.println("String 2");  
                break;  
        }  
    }  
}
```

Is this space-efficient? Time-efficient? How is it implemented?

```

26: invokevirtual #2      // Method java/lang/String.hashCode:()I
29: lookupswitch {
    96354: 56
    97379: 72
    default: 85
}
56: aload          5
58: ldc            #3      // String abc
60: invokevirtual #4      // Method java/lang/String.equals:(Ljava/lang/Object;)Z
63: ifeq          85
66: iconst_0
67: istore         6
69: goto          85

72: aload          5
74: ldc            #5      // String bde
76: invokevirtual #4      // Method java/lang/String.equals:(Ljava/lang/Object;)Z
79: ifeq          85
82: iconst_1
83: istore         6

85: iload          6
87: lookupswitch {
    0: 112
    1: 123
    default: 131
}
112: getstatic     #6      // Field java/lang/System.out:Ljava/io/PrintStream;
115: ldc            #7      // String String 1
117: invokevirtual #8      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
120: goto          131

123: getstatic     #6      // Field java/lang/System.out:Ljava/io/PrintStream;
126: ldc            #9      // String String 2
128: invokevirtual #8      // Method java/io/PrintStream.println:(Ljava/lang/String;)V
131: ...

```

```
switch (s) {  
  case "abc":  
    System.out.println("String 1");  
    break;  
  case "bde":  
    System.out.println("String 2");  
    break;  
}
```



```
switch (s) {  
  case "abc":  
    System.out.println("String 1");  
    break;  
  case "bde":  
    System.out.println("String 2");  
    break;  
}
```

```
// String s1  
byte byte0 = -1;  
switch (s1.hashCode()) {  
  case 96354: if (s1.equals("abc")) byte0 = 0; break;  
  case 97379: if (s1.equals("bde")) byte0 = 1; break;  
}  
  
switch (byte0) {  
  case 0: System.out.println("String 1"); break;  
  case 1: System.out.println("String 2"); break;  
}
```

Multicatches

```
public void ioexception() throws IOException;
public void runtimeexception() throws RuntimeException;
public void urisyntaxexception() throws URISyntaxException;

public void example1() {
    try {
        ioexception();
        runtimeexception();
        urisyntaxexception();
    } catch (IOException | RuntimeException |
             URISyntaxException e) {
        System.out.println("Caught: " + e);
    } finally {
        System.out.println("Finally.");
    }
}
```

Multicatches

```
public void ioexception() throws IOException;
public void runtimeexception() throws RuntimeException;
public void urisyntaxexception() throws URISyntaxException;

public void example1() {
    try {
        ioexception();
        runtimeexception();
        urisyntaxexception();
    } catch (IOException | RuntimeException |
             URISyntaxException e) {
        System.out.println("Caught: " + e);
    } finally {
        System.out.println("Finally.");
    }
}
```

What's the benefit over multiple-catch statements?

```

0: invokestatic #9 // Method ioexception:()V
3: invokestatic #10 // Method runtimeexception:()V
6: invokestatic #11 // Method urisyntaxexception:()V
9: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
12: ldc #13 // String Finally.
14: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
17: goto 68

20: astore_0
21: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
24: new #15 // class java/lang/StringBuilder
27: dup
28: invokespecial #16 // Method java/lang/StringBuilder.<init>:()V
31: ldc #17 // String Caught:
33: invokevirtual #18 // Method java/lang/StringBuilder.append:(Ljava/l
36: aload_0
37: invokevirtual #19 // Method java/lang/StringBuilder.append:(Ljava/l
40: invokevirtual #20 // Method java/lang/StringBuilder.toString:()Ljav
43: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
46: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
49: ldc #13 // String Finally.
51: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
54: goto 68
57: astore_1
58: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
61: ldc #13 // String Finally.
63: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
66: aload_1
67: athrow
68: return

```

Exception table:

from	to	target	type
0	9	20	Class java/io/IOException
0	9	20	Class java/lang/RuntimeException
0	9	20	Class java/net/URISyntaxException
0	9	57	any
20	46	57	any
57	58	57	any

```

0: invokestatic #9 // Method ioexception:()V
3: invokestatic #10 // Method runtimeexception:()V
6: invokestatic #11 // Method urisyntaxexception:()V
9: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
12: ldc #13 // String Finally.
14: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
17: goto 68

• 20: astore_0
21: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
24: new #15 // class java/lang/StringBuilder
27: dup
28: invokespecial #16 // Method java/lang/StringBuilder.<init>:()V
31: ldc #17 // String Caught:
33: invokevirtual #18 // Method java/lang/StringBuilder.append:(Ljava/l
36: aload_0
37: invokevirtual #19 // Method java/lang/StringBuilder.append:(Ljava/l
40: invokevirtual #20 // Method java/lang/StringBuilder.toString:()Ljav
43: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
46: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
49: ldc #13 // String Finally.
51: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
54: goto 68

• 57: astore_1
58: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
61: ldc #13 // String Finally.
63: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang
66: aload_1
67: athrow
• 68: return

```

Exception table:

from	to	target	type
0	9	20	Class java/io/IOException
0	9	20	Class java/lang/RuntimeException
0	9	20	Class java/net/URISyntaxException
0	9	57	any
20	46	57	any
57	58	57	any

More precise rethrow (?)

```
public void ioexception() throws IOException;
public void runtimeexception() throws RuntimeException;
public void urisyntaxexception() throws URISyntaxException;

public static void example2()
    throws IOException,
        URISyntaxException {
    try {
        ioexception();
        runtimeexception();
        urisyntaxexception();
    } catch (Exception e) {
        System.out.println("Caught: " + e);
        throw e;
    }
}
```

More precise rethrow (?)

```
public void ioexception() throws IOException;
public void runtimeexception() throws RuntimeException;
public void urisyntaxexception() throws URISyntaxException;

public static void example2()
    throws IOException,
        URISyntaxException {
    try {
        ioexception();
        runtimeexception();
        urisyntaxexception();
    } catch (Exception e) {
        System.out.println("Caught: " + e);
        throw e;
    }
}
```

At the JVM level method exception signatures are a no-op.

Code:

```
0: invokestatic #9 // Method ioexception:()V
3: invokestatic #10 // Method runtimeexception:()V
6: invokestatic #11 // Method urisyntaxexception:()V
9: goto 40

12: astore_0
13: getstatic #12 // Field java/lang/System.out:Ljava/io/PrintStream;
16: new #15 // class java/lang/StringBuilder
19: dup
20: invokespecial #16 // Method java/lang/StringBuilder."<init>":()V
23: ldc #17 // String Caught:
25: invokevirtual #18 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava
28: aload_0
29: invokevirtual #19 // Method java/lang/StringBuilder.append:(Ljava/lang/Object;)Ljava
32: invokevirtual #20 // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
35: invokevirtual #14 // Method java/io/PrintStream.println:(Ljava/lang/String;)V
38: aload_0
39: athrow

40: return
```

Exception table:

from	to	target type
0	9	12 Class java/lang/Exception

Try-With-Resource

```
static class MyResource implements AutoCloseable {  
    public void doSomething() {}  
  
    public void close() throws Exception {  
        // Close me!  
    }  
}  
  
public static void main(String [] args) throws Exception {  
    try (MyResource resource = new MyResource()) {  
        resource.doSomething();  
    }  
}
```

Try-With-Resource

```
static class MyResource implements AutoCloseable {  
    public void doSomething() {}  
  
    public void close() throws Exception {  
        // Close me!  
    }  
}  
  
public static void main(String [] args) throws Exception {  
    try (MyResource resource = new MyResource()) {  
        resource.doSomething();  
    }  
}
```

How would you accomplish this manually?

Poor man's resource cleanup:

```
final MyResource resource = new MyResource();
try {
    resource.doSomething();
    ...
} finally {
    try {
        resource.close();
    } catch (Exception e) {
        // Ignore, not much to do.
    }
}
```

Java 7 generated resource cleanup:

```
Throwable blockError = null;
final MyResource resource = /* expr */ new MyResource();
try {
    resource.doSomething();
} catch (Throwable t) {
    blockError = t;
    throw blockError;
} finally {
    if (resource != null) {
        if (blockError != null) {
            try {
                resource.close();
            } catch (Throwable t) {
                blockError.addSuppressed(t);
            }
        } else {
            resource.close();
        }
    }
}
```

Java 7 generated resource cleanup:

```
Throwable blockError = null;
final MyResource resource = /* expr */ new MyResource();
try {
    resource.doSomething();
} catch (Throwable t) {
    blockError = t;
    throw blockError;
} finally {
    if (resource != null) {
        if (blockError != null) {
            try {
                resource.close();
            } catch (Throwable t) {
                blockError.addSuppressed(t);
            }
        } else {
            resource.close();
        }
    }
}
```

Now you can explain why resource is final inside the block!

try... with handlers

```
try (MyResource resource = new MyResource()) {  
    resource.doSomething();  
} catch (Exception e) {  
    ... // resource not visible => closed  
} finally {  
    ... // resource not visible => closed  
}
```

try... with handlers

```
try (MyResource resource = new MyResource()) {  
    resource.doSomething();  
} catch (Exception e) {  
    ... // resource not visible => closed  
} finally {  
    ... // resource not visible => closed  
}
```

How is this unrolled?

try... with handlers

```
try {  
    try (MyResource resource = new MyResource()) {  
        resource.doSomething();  
    };  
} catch (Exception e) {  
    ...  
} finally {  
    ...  
}
```


HotSpot and GC improvements

The default GC (G1GC?)

```
for (GarbageCollectorMXBean i :  
    ManagementFactory.getGarbageCollectorMXBeans()) {  
    System.out.println(i.getName());  
}
```

The default GC (G1GC?)

```
for (GarbageCollectorMXBean i :  
    ManagementFactory.getGarbageCollectorMXBeans()) {  
    System.out.println(i.getName());  
}
```

PS Scavenge
PS MarkSweep

Seems to be -XX:+UseParallelGC.

The default GC (G1GC?)

```
for (GarbageCollectorMXBean i :  
    ManagementFactory.getGarbageCollectorMXBeans()) {  
    System.out.println(i.getName());  
}
```

PS Scavenge

PS MarkSweep

The default GC (G1GC?)

```
for (GarbageCollectorMXBean i :  
    ManagementFactory.getGarbageCollectorMXBeans()) {  
    System.out.println(i.getName());  
}
```

PS Scavenge
PS MarkSweep

**But G1GC is available without diagnostic switches:
-XX:+UseG1GC**

G1 Young Generation
G1 Old Generation

The default jit compiler

```
System.out.println(  
    ManagementFactory.getCompilationMXBean().getName());
```

The default jit compiler

```
System.out.println(  
    ManagementFactory.getCompilationMXBean().getName());
```

HotSpot 64-Bit Tiered Compilers

Any new intrinsics?

Diffing openjdk code.

vmSymbols.hpp

Any new intrinsics?

Diffing openjdk code.

vmSymbols.hpp

No differences to 1.6u26.

Other than invokedynamic. So no advantage over 1.6 here.

invokedynamic

Duck-typing

But allowing jit optimizations.

java.lang.invoke

Package changed from previous java.dyn.

In practice: no opinion.

Not much to be told (yet).

New jit, new bugs

As reported in Lucene (LUCENE-3335):

```
curl http://tartarus.org/~martin/PorterStemmer/java.txt > Stemmer.java
javac Stemmer.java
java Stemmer /usr/share/dict/words
```

New jit, new bugs

As reported in Lucene (LUCENE-3335):

```
curl http://tartarus.org/~martin/PorterStemmer/java.txt > Stemmer.java
javac Stemmer.java
java Stemmer /usr/share/dict/words
```

...

maisi

maisi's

maitreya

maitreya's

#

A fatal error has been detected by the Java Runtime Environment:

#

EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000025c6f7a, pid=3852, tid=1492

#

JRE version: 7.0-b147

Java VM: Java HotSpot(TM) 64-Bit Server VM (21.0-b17 mixed mode windows-amd64 compressed o

Problematic frame:

J Stemmer.step4()V

Benchmarking experiments

Morfologik, FSA5

Repeated construction of automata.

Linux and Windows, two different processors.

Mostly CPU-bound.

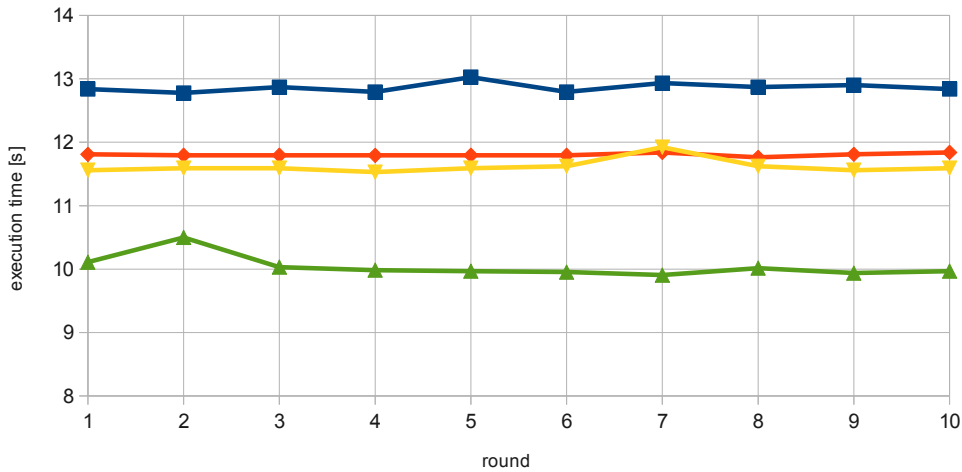
Although large heap required.

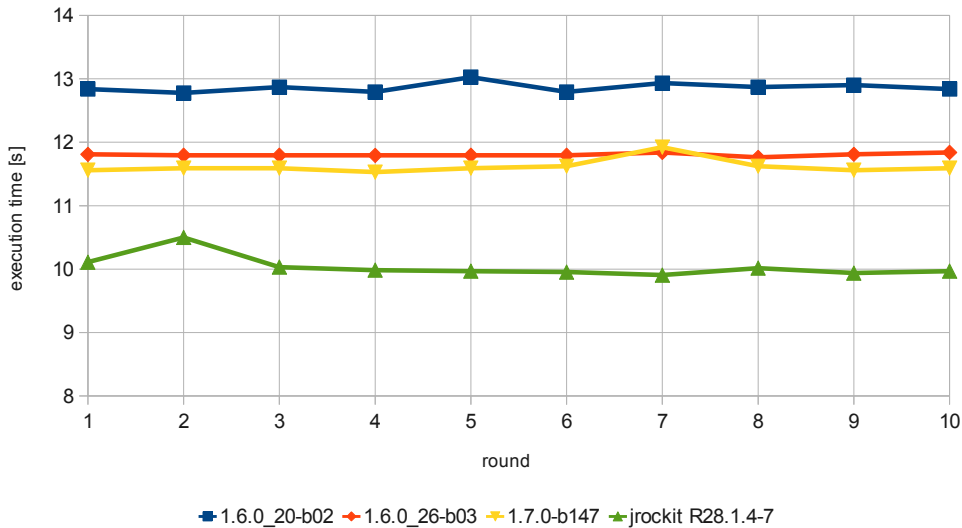
Swap turned off.

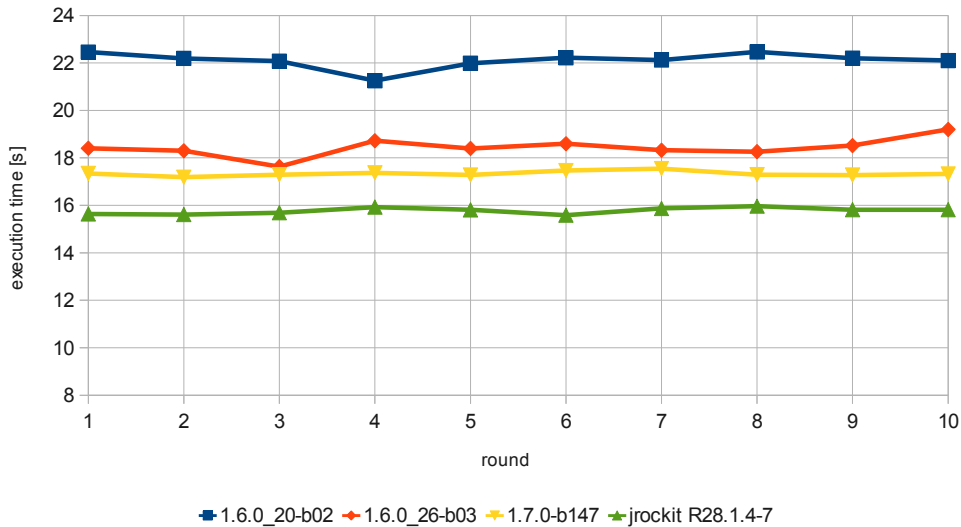
Linux only, nothing done on Windows.

Four JVMs.

HotSpot 1.6_u20, HotSpot 1.6_u26, 1.7pre, jrokit R28.1.4-7.







Lucene FST

Repeated construction of automata.

Linux and Windows, two different processors.

Mostly CPU-bound.

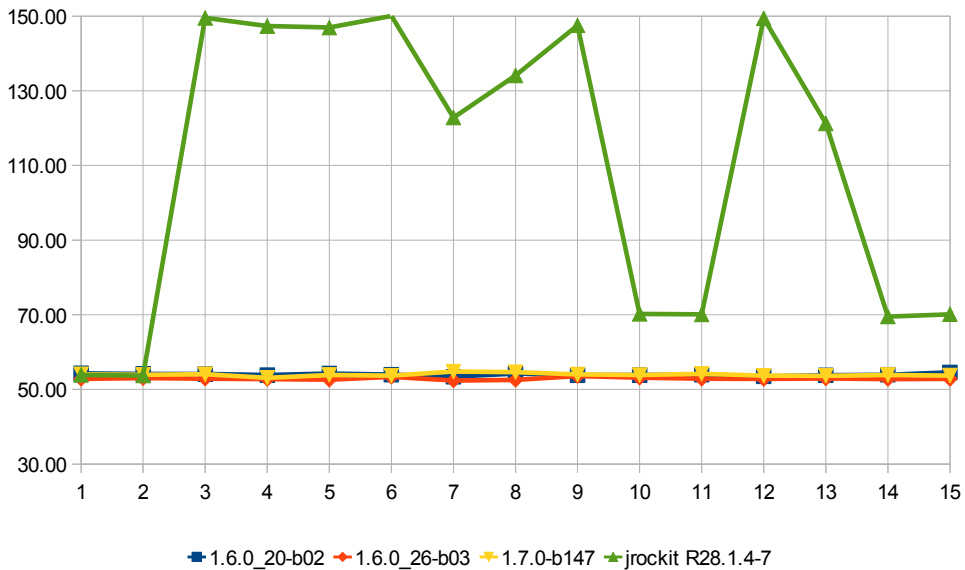
Although some GC activity in place.

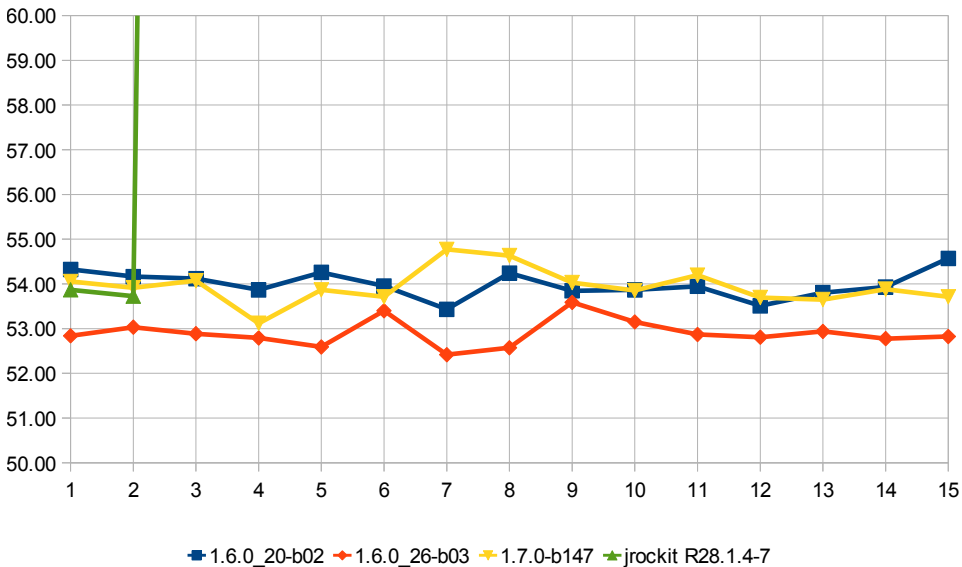
Swap turned off.

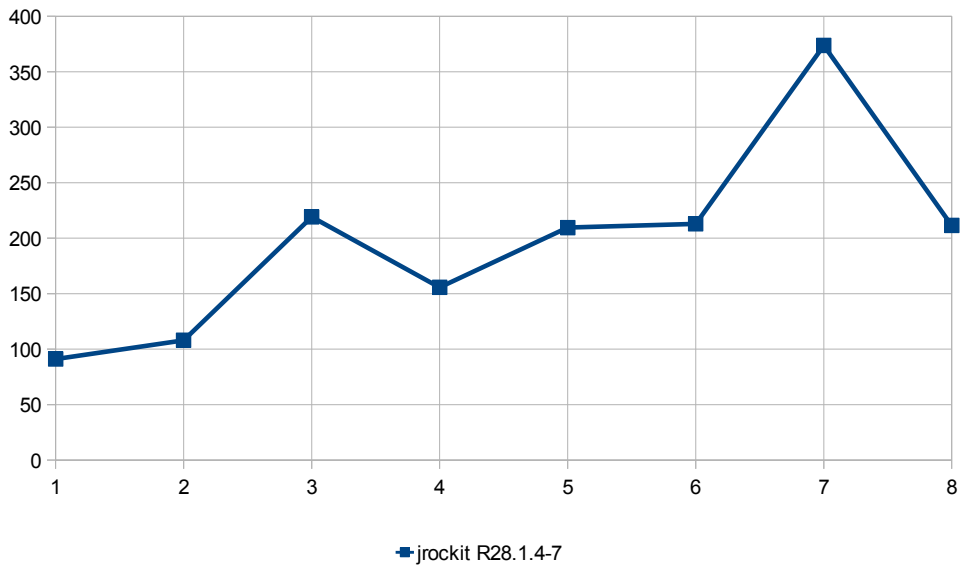
Linux only, nothing done on Windows.

Four JVMs.

HotSpot 1.6_u20, HotSpot 1.6_u26, 1.7pre, jrockit R28.1.4-7.







Conclusions

Conclusions

Nice syntactic sugar improvements.

...but far from revolutionary.

Not much in terms of jit performance.

Definitely a good step forward.