拉勾教育
—互联网人实战大学—

《Java性能优化与面试21讲》

李国

— 拉勾教育出品 —

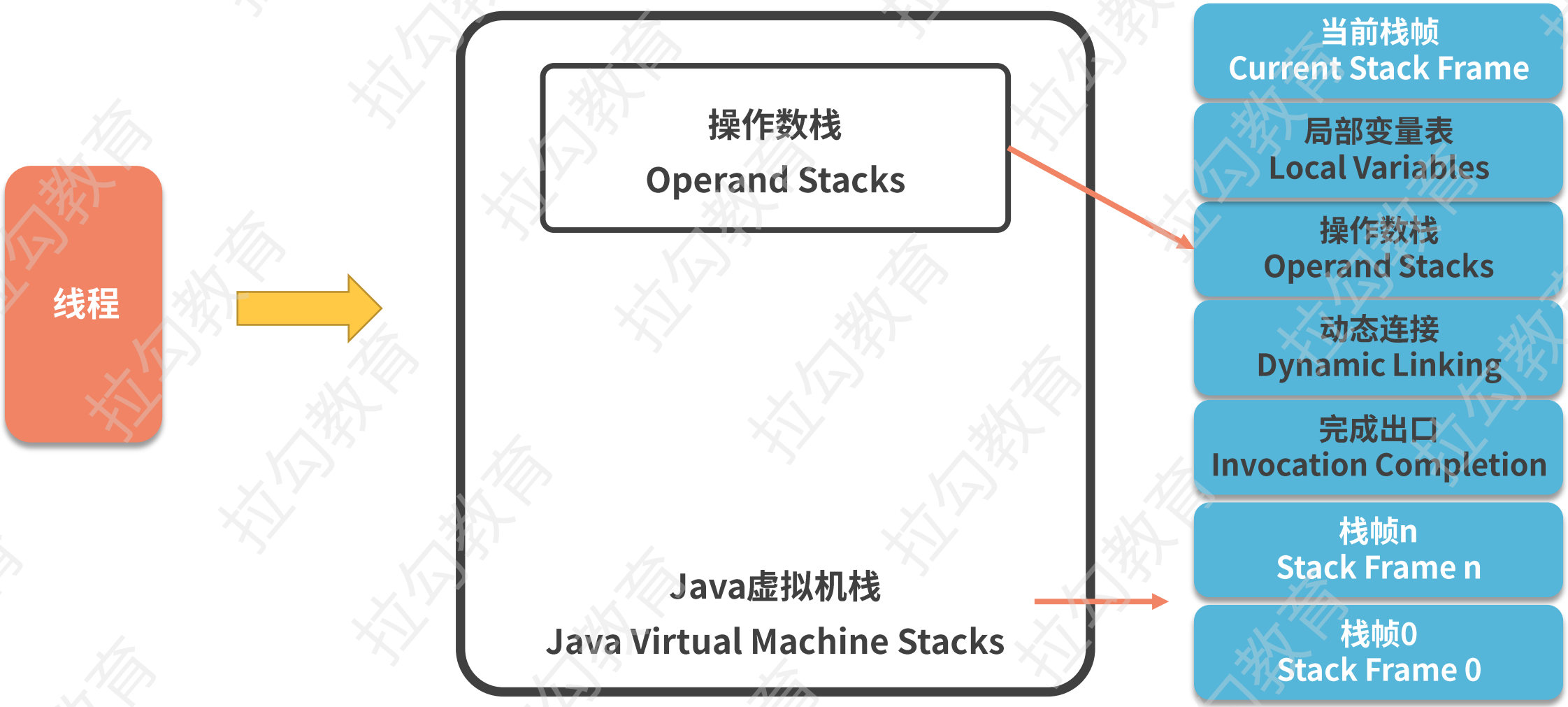
拉勾教育

拉勾教育
—互联网人实战大学—

18｜高级进阶：JIT如何影响JVM的性能？

L / A / G / O / U

前言


线程
操作数栈
Operand Stacks
Java虚拟机栈
Java Virtual Machine Stacks
当前栈帧
Current Stack Frame
局部变量表
Local Variables
操作数栈
Operand Stacks
动态连接
Dynamic Linking
完成出口
Invocation Completion
栈帧n
Stack Frame n
栈帧0
Stack Frame 0

前言

拉勾教育
互联网人实战大学

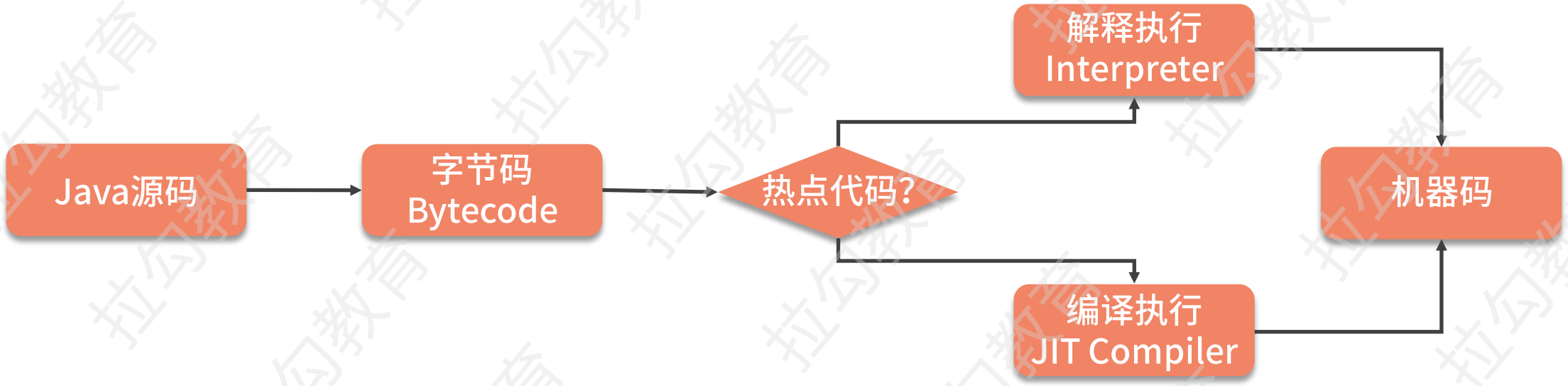
```
public class A{
    int attr = 0;
    public void test(){
        int a = attr;
        System.out.println("ok");
    }
}
```

拉勾教育

前言


```
public void test();
  descriptor: ()V
  flags: ACC_PUBLIC
  Code:
    stack=2, locals=2, args_size=1
       0: aload_0
       1: getfield      #2                  // Field attr:I
       4: istore_1
       5: getstatic     #3                  // Field
java/lang/System.out:Ljava/io/PrintStream;
       8: ldc           #4                  // String ok
      10: invokevirtual #5                  // Method
java/io/PrintStream.println:(Ljava/lang/String;)V
      13: return
    LineNumberTable:
      line 4: 0
      line 5: 5
      line 6: 13
```

前言


Java源码
字节码
Bytecode
热点代码?
解释执行
Interpreter
编译执行
JIT Compiler
机器码

方法内联

拉勾教育
互联网人实战大学

inline——内联

会把一些短小的方法体，直接纳入到目标方法的作用范围之内

就像是直接在代码块中追加代码

方法内联

拉勾教育
互联网人实战大学

使用 -XX:-Inline 参数来禁用方法内联

如果想要更细粒度的控制，可以使用 CompileCommand 参数

```
-XX:CompileCommand=exclude,java/lang/String.indexOf
```

方法内联

拉勾教育
互联网人实战大学

```
public void target_blank() {
    // this method was intentionally left blank
}

@CompilerControl(CompilerControl.Mode.DONT_INLINE)
public void target_dontInline() {
    // this method was intentionally left blank
}


@CompilerControl(CompilerControl.Mode.INLINE)
public void target_inline() {
    // this method was intentionally left blank
}

@CompilerControl(CompilerControl.Mode.EXCLUDE)
public void target_exclude() {
    // this method was intentionally left blank
}
```

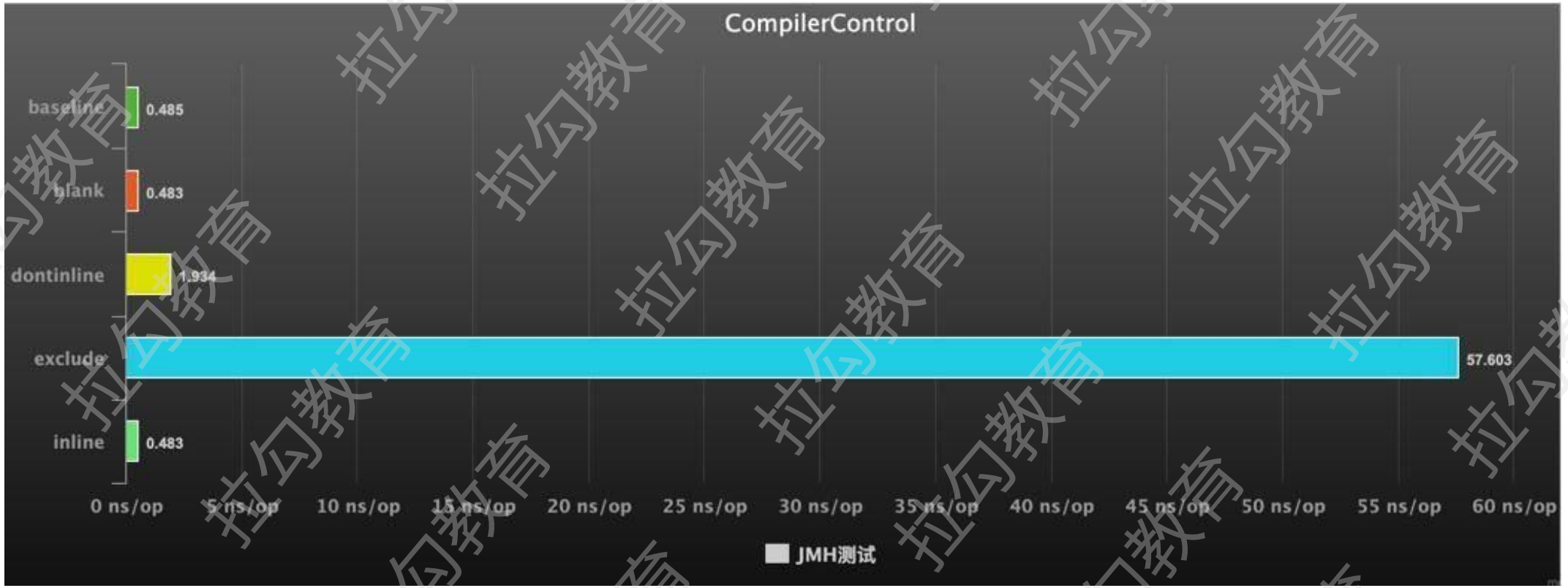
L / A / G / O / U

方法内联


```
Benchmark                                  Mode  Cnt   Score   Error  Units
JMHSample_16_CompilerControl.baseline      avgt    3   0.485 ± 1.492  ns/op
JMHSample_16_CompilerControl.blank         avgt    3   0.483 ± 1.518  ns/op
JMHSample_16_CompilerControl.dontinline    avgt    3   1.934 ± 3.112  ns/op
JMHSample_16_CompilerControl.exclude       avgt    3  57.603 ± 4.435  ns/op
JMHSample_16_CompilerControl.inline        avgt    3   0.483 ± 1.520  ns/op
```

方法内联

拉勾教育
互联网人实战大学



编译层次


```
"C2 CompilerThread0" #6 daemon prio=9 os_prio=31 cpu=830.41ms
elapsed=4252.14s tid=0x00007ffaed023000 nid=0x5a03 waiting on condition
[0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
   No compile task

"C1 CompilerThread0" #8 daemon prio=9 os_prio=31 cpu=549.91ms
elapsed=4252.14s tid=0x00007ffaed831800 nid=0x5c03 waiting on condition
[0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
   No compile task
```

编译层次

拉勾教育
互联网人实战大学

01 字节码的解释执行

02 执行不带 profiling 的 C1 代码

03 执行仅带方法调用次数以及循环执行次数 profiling 的 C1 代码

04 执行带所有 profiling 的 C1 代码

05 执行 C2 代码

逃逸分析

拉勾教育
互联网人实战大学

~~对象，除了基本数据类型，一定是在堆上分配的么？~~

逃逸分析


```
public class EscapeAttr {
    Object attr;
    public void test() {
        attr = new Object();
    }
}
```

逃逸分析


```
public class EscapeReturn {
    Object attr;
    public Object test() {
        Object obj = new Object();
        return obj;
    }
}
```

逃逸分析

逃逸分析

同步消除

```
public class SyncEliminate{
    public void test() {
        synchronized (new Object()){
        }
    }
}
```

拉勾教育

逃逸分析

同步消除

```
Benchmark                                  Mode  Cnt      Score      Error  Units
BuilderVsBufferBenchmark.buffer  thrpt   10  90085.927 ± 95174.289  ops/ms
BuilderVsBufferBenchmark.builder thrpt   10  103280.200 ± 76172.538  ops/ms
```

JITWatch

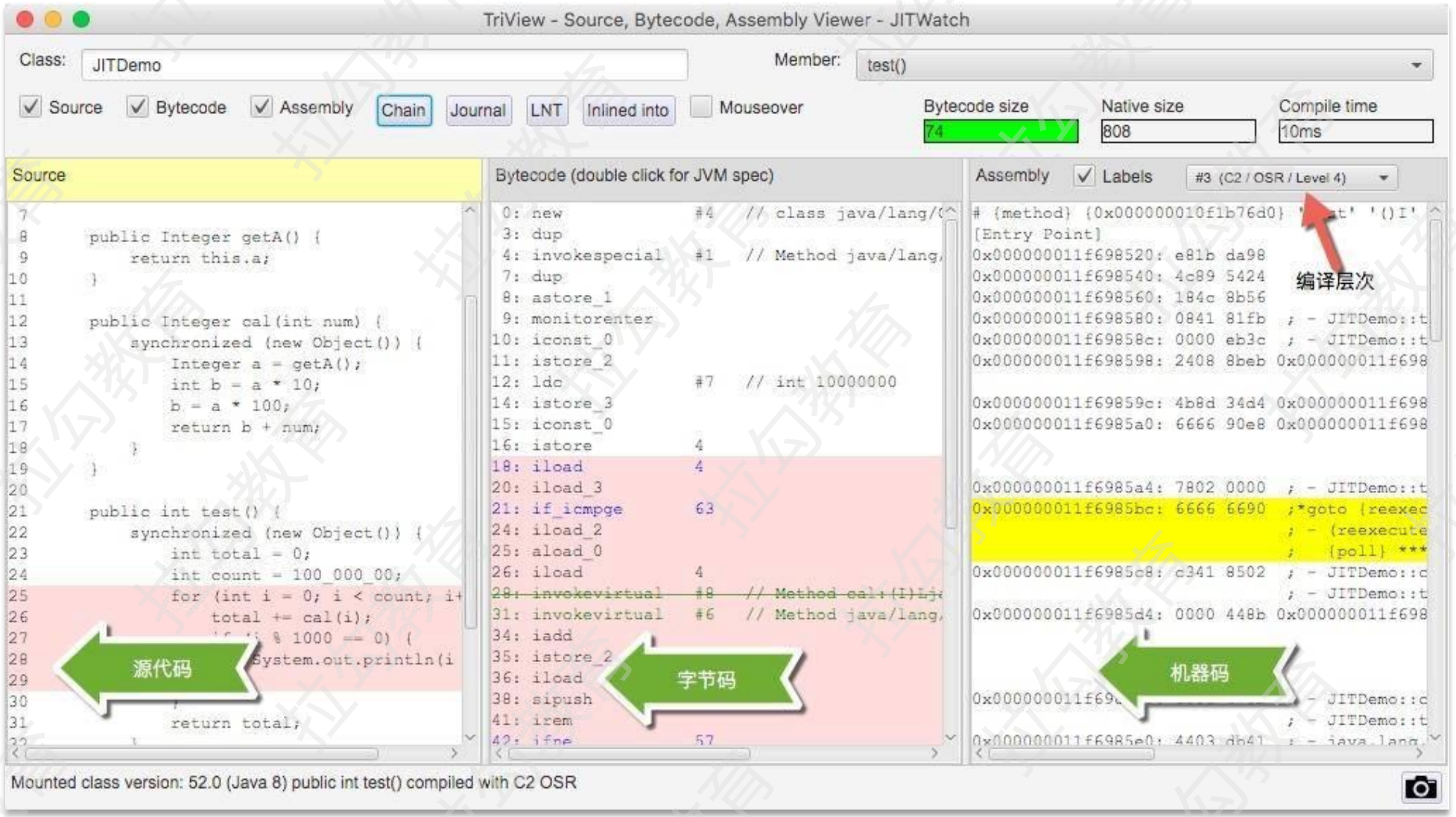
可以使用 jitwatch 工具来观测 JIT 的一些行为

https://github.com/AdoptOpenJDK/jitwatch

JITWatch


```
-XX:+UnlockDiagnosticVMOptions -XX:+TraceClassLoading -
XX:+PrintAssembly -XX:+LogCompilation -XX:LogFile=jitdemo.log
```

JITWatch


TriView - Source, Bytecode, Assembly Viewer - JITWatch
Class: JITDemo
Member: test()
Source
Bytecode
Assembly
Chain
Journal
LNT
Inlined into
Mouseover
Bytecode size
Native size
Compile time
74
808
10ms
Source
Bytecode (double click for JVM spec)
Assembly
Labels
#3 (C2 / OSR / Level 4)
编译层次
源代码
字节码
机器码
Mounted class version: 52.0 (Java 8) public int test() compiled with C2 OSR

JITWatch

拉勾教育
互联网人实战大学

```
public class SimpleInliningTest{
    public SimpleInliningTest() {
        int sum = 0;
        // 1_000_000 is F4240 in hex
        for (int i = 0; i < 1_000_000; i++) {
            sum = this.add(sum, 99);
            // 63 hex
        }
        System.out.println("Sum:" + sum);
    }

    public int add(int a, int b) {
        return a + b;
    }
    public static void main(String[] args) {
        new SimpleInliningTest();
    }
}
```

拉勾教育

JITWatch

拉勾教育
互联网人实战大学

```
box is designed to help you learn about the HotSpot JIT compilers.
ote that the JIT compilers may behave differently when isolating a method
andbox compared to running your whole application.

s SimpleInliningTest

SimpleInliningTest()

 sum = 0;

1_000_000 is F4240 in hex
 (int i = 0 ; i < 1_000_000; i++)

 sum = this.add(sum, 99); // 63 hex
```

```
 0: aload_0
 1: invokespecial   #1   // Method java/lang/Object."<init>":()V
 4: iconst_0
 5: istore_1
 6: iconst_0
 7: istore_2
 8: iload_2
 9: ldc             #2   // int 1000000
11: if_icmpge       28
14: aload_0
15: iload_1
16: bipush          99
18: invokevirtual   #3   // Method add:(II)I
21: istore_1
22: iinc            2, 1
25: goto            8
```

Inlining report for callee public int add(int,int)

Filter on package prefixes (comma separated)

Caller Class	Caller method	Compilation	BCI	Inlined?	Reason
SimpleInliningTest	SimpleInliningTest()	#1 (C1 / OSR / Level 3)	View BCI 18	Yes	Yes, receiver is statically known
SimpleInliningTest	SimpleInliningTest()	#2 (C1 / Level 3)	View BCI 18	Yes	Yes, receiver is statically known
SimpleInliningTest	SimpleInliningTest()	#3 (C2 / OSR / Level 4)	View BCI 18	Yes	Yes, inline (hot)

小结

拉勾教育
互联网人实战大学

JIT 是现代 JVM 主要的优化点，能够显著地提升程序的执行效率

JIT 在某些情况下还会出现逆优化

小结

拉勾教育
互联网人实战大学

```
public class Demo {
    static final class TestThread extends Thread {
        boolean stop = false;

        public boolean isStop() {
            return stop;
        }

        @Override
        public void run() {
            try {
                Thread.sleep(100);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            stop = true;
            System.out.println("END");
        }
    }
```

LAGOU

小结


```
            Thread.sleep(100);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        stop = true;
        System.out.println("END");
    }
}

public static void main(String[] args) {
    int i = 0;
    TestThread test = new TestThread();
    test.start();
    while(!test.isStop()) {
        System.out.println("--");
        i++;
    }
  }
}
```

拉勾教育

Next：第19讲《高级进阶：JVM 常见优化参数》

拉勾教育
—互联网人实战大学—
L / A / G / O / U
下载「拉勾教育App」
获取更多内容