JVM如何实现invokedynamic？

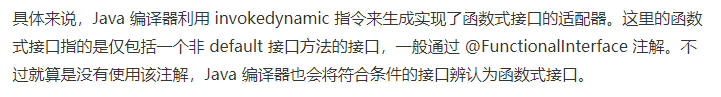
讲讲如何生成调用点，并且启用程序自己决定链接至哪个方法。

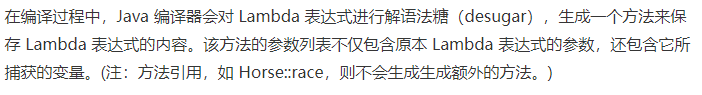
## Invokeddynamic指令

Java7中引入的一条新的指令，用以支持动态语言的方法调用。将调用点（CallSite）抽象成为Java类，

## Java8Lambda表达式

Java8中，Lambda表达式也是借助invokedynamic实现的。

具体来说，java编译器利用invokedynamic指令来生成了函数式接口的适配器。



## -Djdk.internal.lambda.dumpProxyClasses=/DUMP/PATH

**int** x =1;  
IntStream intStream = IntStream.of(1, 2, 3).map(i -> i \* 2).map(i -> i \* x);  
**System**.out.println(**Arrays**.toString(intStream.toArray()));

导出具体的适配器类

// i->i\*2 对应的适配器类

final class LambdaTest$$Lambda$1 implements IntUnaryOperator {

private LambdaTest$$Lambda$1();

Code:

0: aload\_0

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

4: return

public int applyAsInt(int);

Code:

0: iload\_1

1: invokestatic LambdaTest.lambda$0:(I)I

4: ireturn

}

// i->i\*x 对应的适配器类

final class LambdaTest$$Lambda$2 implements IntUnaryOperator {

private final int arg$1;

private LambdaTest$$Lambda$2(int);

Code:

0: aload\_0

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

4: aload\_0

5: iload\_1

6: putfield arg$1:I

9: return

private static java.util.function.IntUnaryOperator get$Lambda(int);

Code:

0: new LambdaTest$$Lambda$2

3: dup

4: iload\_0

5: invokespecial "<init>":(I)V

8: areturn

public int applyAsInt(int);

Code:

0: aload\_0

1: getfield arg$1:I

4: iload\_1

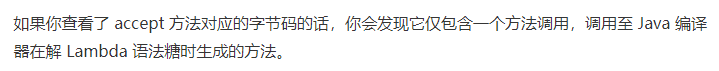
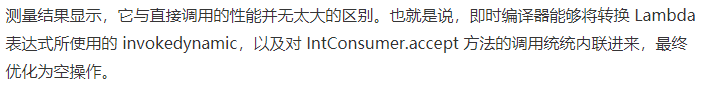
5: invokestatic LambdaTest.lambda$1:(II)I

8: ireturn

}

## Lambda以及方法句柄性能分析

**long** cur = **System**.currentTimeMillis();  
 **for** (**int** i =0; i <= 2\_000\_000\_000; i ++) {  
 **if** (i % 100\_000\_000 == 0) {  
 **long** tem = **System**.currentTimeMillis();  
 **System**.out.println(tem - cur);  
 cur = tem;  
 }  
  
// ((IntConsumer) Test::target).accept(128);  
// target(128);  
 ((IntConsumer) j -> **Test**.target(j)).accept(128);  
 }



## DoEscapeAnalysis

设置逃逸分析的开关。逃逸分析能够去除额外的新建实例的开销，但是也不是时刻奏效。需要同时满足两件事：

1. invokedynamic指令所执行的方法句柄能够内联，
2. 和接下来的accept方法调用也能内联

只有这样，逃逸分析才会判断该适配器实例不逃逸。否则，我们会在运行的过程中不停的生成适配器实例。所以，我们应当尽量设置使用非捕获的Lambda表达式。

捕获Lambda表达式借助了及时编译器中的逃逸分析，来避免实际的新建适配器类实例的操作。