## Lambda实现原理

先从一个例子开始

***public class LambdaTest {***

***public static void print(String name, Print print){***

***print.print(name);***

***}***

***public static void main(String [] args) {***

***String name = "Chen Longfei";***

***String prefix = "hello, ";***

***print(name, (t) -> System.out.println(t));***

***//与上一行不同的是，Lambda表达式的函数体中引用了外部变量‘prefix’***

***print(name, (t) -> System.out.println(prefix + t));***

***}***

***}***

***@FunctionalInterface***

***interface Print {***

***void print(String name);***

***}***

例子很简单，定义了一个函数式接口***Print*** ，main方法中有两处代码以Lambda表达式的方式实现了print接口，分别打印出不带前缀与带前缀的名字。

运行程序，打印结果如下：

***Chen Longfei***

***hello, Chen Longfei***

而(t) -> System.out.println(t)与(t) -> System.out.println(prefix + t))之类的Lambda表达式到底是怎样被编译与调用的呢？

我们知道，编译器编译Java代码时经常在背地里“搞鬼”比如类的全限定名的补全，泛型的类型推断等，编译器耍的这些小聪明可以帮助我们写出更优雅、简洁、高效的代码。鉴于编译器的一贯作风，我们有理由怀疑，新颖而另类的Lambda表达式在编译时很可能会被改造过了。

下面通过javap反编译class文件一探究竟。

Javap是jdk自带的一个字节码查看工具及反编译工具：

***用法: javap <options> <classes>***

***其中, 可能的选项包括:***

***-help --help -? 输出此用法消息***

***-version 版本信息***

***-v -verbose 输出附加信息***

***-l 输出行号和本地变量表***

***-public 仅显示公共类和成员***

***-protected 显示受保护的/公共类和成员***

***-package 显示程序包/受保护的/公共类***

***和成员 (默认)***

***-p -private 显示所有类和成员***

***-c 对代码进行反汇编***

***-s 输出内部类型签名***

***-sysinfo 显示正在处理的类的***

***系统信息 (路径, 大小, 日期, MD5 散列)***

***-constants 显示最终常量***

***-classpath <path> 指定查找用户类文件的位置***

***-cp <path> 指定查找用户类文件的位置***

***-bootclasspath <path> 覆盖引导类文件的位置***

javap -p Print.class

结果如下：

interface test.Print {

public abstract void print(java.lang.String);

}

javap -p LambdaTest.class

结果如下：

Compiled from "LambdaTest.java"

public class test.LambdaTest {

public test.LambdaTest();

public static void print(java.lang.String, test.Print);

public static void main(java.lang.String[]);

private static void lambda$main$1(java.lang.String);

private static void lambda$main$0(java.lang.String, java.lang.String);

}

可见，编译器对Print接口的改造比较小，只是为print方法添加了public abstract关键字，而对LambdaTest的变化就比较大了，添加了两个静态方法：

* private static void lambda$main$1(java.lang.String);
* private static void lambda$main$0(java.lang.String, java.lang.String);

对比原生的java代码，很容易做出推测，这两个静态方法与两处Lambda表达式相关：

print(name, (t) -> System.out.println(t));

print(name, (t) -> System.out.println(prefix + t));

到底有什么关联呢？使用javap -p -v -c LambdaTest.class查看更加详细的反编译结果：

public class test.LambdaTest

minor version: 0

major version: 52

flags: ACC\_PUBLIC, ACC\_SUPER

Constant pool:

#1 = Methodref #15.#30 // java/lang/Object."<init>":()V

#2 = InterfaceMethodref #31.#32 // test/Print.print:(Ljava/lang/String;)V

#3 = String #33 // Chen Longfei

#4 = String #34 // hello,

#5 = InvokeDynamic #0:#39 // #0:print:(Ljava/lang/String;)Ltest/Print;

#6 = Methodref #14.#40 // test/LambdaTest.print:(Ljava/lang/String;Ltest/Print;)V

#7 = InvokeDynamic #1:#42 // #1:print:()Ltest/Print;

#8 = Fieldref #43.#44 // java/lang/System.out:Ljava/io/PrintStream;

#9 = Methodref #45.#46 // java/io/PrintStream.println:(Ljava/lang/String;)V

#10 = Class #47 // java/lang/StringBuilder

#11 = Methodref #10.#30 // java/lang/StringBuilder."<init>":()V

#12 = Methodref #10.#48 // java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder

;

#13 = Methodref #10.#49 // java/lang/StringBuilder.toString:()Ljava/lang/String;

#14 = Class #50 // test/LambdaTest

#15 = Class #51 // java/lang/Object

#16 = Utf8 <init>

#17 = Utf8 ()V

#18 = Utf8 Code

#19 = Utf8 LineNumberTable

#20 = Utf8 print

#21 = Utf8 (Ljava/lang/String;Ltest/Print;)V

#22 = Utf8 main

#23 = Utf8 ([Ljava/lang/String;)V

#24 = Utf8 lambda$main$1

#25 = Utf8 (Ljava/lang/String;)V

#26 = Utf8 lambda$main$0

#27 = Utf8 (Ljava/lang/String;Ljava/lang/String;)V

#28 = Utf8 SourceFile

#29 = Utf8 LambdaTest.java

#30 = NameAndType #16:#17 // "<init>":()V

#31 = Class #52 // test/Print

#32 = NameAndType #20:#25 // print:(Ljava/lang/String;)V

#33 = Utf8 Chen Longfei

#34 = Utf8 hello,

#35 = Utf8 BootstrapMethods

#36 = MethodHandle #6:#53 // invokestatic java/lang/invoke/LambdaMetafactory.metafactory:(Ljava/lang/inv

oke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/M

ethodHandle;Ljava/lang/invoke/MethodType;)Ljava/lang/invoke/CallSite;

#37 = MethodType #25 // (Ljava/lang/String;)V

#38 = MethodHandle #6:#54 // invokestatic test/LambdaTest.lambda$main$0:(Ljava/lang/String;Ljava/lang/St

ring;)V

#39 = NameAndType #20:#55 // print:(Ljava/lang/String;)Ltest/Print;

#40 = NameAndType #20:#21 // print:(Ljava/lang/String;Ltest/Print;)V

#41 = MethodHandle #6:#56 // invokestatic test/LambdaTest.lambda$main$1:(Ljava/lang/String;)V

#42 = NameAndType #20:#57 // print:()Ltest/Print;

#43 = Class #58 // java/lang/System

#44 = NameAndType #59:#60 // out:Ljava/io/PrintStream;

#45 = Class #61 // java/io/PrintStream

#46 = NameAndType #62:#25 // println:(Ljava/lang/String;)V

#47 = Utf8 java/lang/StringBuilder

#48 = NameAndType #63:#64 // append:(Ljava/lang/String;)Ljava/lang/StringBuilder;

#49 = NameAndType #65:#66 // toString:()Ljava/lang/String;

#50 = Utf8 test/LambdaTest

#51 = Utf8 java/lang/Object

#52 = Utf8 test/Print

#53 = Methodref #67.#68 // java/lang/invoke/LambdaMetafactory.metafactory:(Ljava/lang/invoke/MethodHan

dles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodHandle;L

java/lang/invoke/MethodType;)Ljava/lang/invoke/CallSite;

#54 = Methodref #14.#69 // test/LambdaTest.lambda$main$0:(Ljava/lang/String;Ljava/lang/String;)V

#55 = Utf8 (Ljava/lang/String;)Ltest/Print;

#56 = Methodref #14.#70 // test/LambdaTest.lambda$main$1:(Ljava/lang/String;)V

#57 = Utf8 ()Ltest/Print;

#58 = Utf8 java/lang/System

#59 = Utf8 out

#60 = Utf8 Ljava/io/PrintStream;

#61 = Utf8 java/io/PrintStream

#62 = Utf8 println

#63 = Utf8 append

#64 = Utf8 (Ljava/lang/String;)Ljava/lang/StringBuilder;

#65 = Utf8 toString

#66 = Utf8 ()Ljava/lang/String;

#67 = Class #71 // java/lang/invoke/LambdaMetafactory

#68 = NameAndType #72:#76 // metafactory:(Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava

/lang/invoke/MethodType;Ljava/lang/invoke/MethodType;Ljava/lang/invoke/MethodHandle;Ljava/lang/invoke/MethodType;)Ljava/

lang/invoke/CallSite;

#69 = NameAndType #26:#27 // lambda$main$0:(Ljava/lang/String;Ljava/lang/String;)V

#70 = NameAndType #24:#25 // lambda$main$1:(Ljava/lang/String;)V

#71 = Utf8 java/lang/invoke/LambdaMetafactory

#72 = Utf8 metafactory

#73 = Class #78 // java/lang/invoke/MethodHandles$Lookup

#74 = Utf8 Lookup

#75 = Utf8 InnerClasses

#76 = Utf8 (Ljava/lang/invoke/MethodHandles$Lookup;Ljava/lang/String;Ljava/lang/invoke/MethodType;Ljava/

lang/invoke/MethodType;Ljava/lang/invoke/MethodHandle;Ljava/lang/invoke/MethodType;)Ljava/lang/invoke/CallSite;

#77 = Class #79 // java/lang/invoke/MethodHandles

#78 = Utf8 java/lang/invoke/MethodHandles$Lookup

#79 = Utf8 java/lang/invoke/MethodHandles

{

public test.LambdaTest();

descriptor: ()V

flags: ACC\_PUBLIC

Code:

stack=1, locals=1, args\_size=1

0: aload\_0

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

4: return

LineNumberTable:

line 6: 0

public static void print(java.lang.String, test.Print);

descriptor: (Ljava/lang/String;Ltest/Print;)V

flags: ACC\_PUBLIC, ACC\_STATIC

Code:

stack=2, locals=2, args\_size=2

0: aload\_1

1: aload\_0

2: invokeinterface #2, 2 // InterfaceMethod test/Print.print:(Ljava/lang/String;)V

7: return

LineNumberTable:

line 9: 0

line 10: 7

public static void main(java.lang.String[]);

descriptor: ([Ljava/lang/String;)V

flags: ACC\_PUBLIC, ACC\_STATIC

Code:

stack=2, locals=3, args\_size=1

0: ldc #3 // String Chen Longfei

2: astore\_1

3: ldc #4 // String hello,

5: astore\_2

6: aload\_1

7: aload\_2

8: invokedynamic #5, 0 // InvokeDynamic #0:print:(Ljava/lang/String;)Ltest/Print;

13: invokestatic #6 // Method print:(Ljava/lang/String;Ltest/Print;)V

16: aload\_1

17: invokedynamic #7, 0 // InvokeDynamic #1:print:()Ltest/Print;

22: invokestatic #6 // Method print:(Ljava/lang/String;Ltest/Print;)V

25: return

LineNumberTable:

line 13: 0

line 14: 3

line 16: 6

line 18: 16

line 19: 25

private static void lambda$main$1(java.lang.String);

descriptor: (Ljava/lang/String;)V

flags: ACC\_PRIVATE, ACC\_STATIC, ACC\_SYNTHETIC

Code:

stack=2, locals=1, args\_size=1

0: getstatic #8 // Field java/lang/System.out:Ljava/io/PrintStream;

3: aload\_0

4: invokevirtual #9 // Method java/io/PrintStream.println:(Ljava/lang/String;)V

7: return

LineNumberTable:

line 18: 0

private static void lambda$main$0(java.lang.String, java.lang.String);

descriptor: (Ljava/lang/String;Ljava/lang/String;)V

flags: ACC\_PRIVATE, ACC\_STATIC, ACC\_SYNTHETIC

Code:

stack=3, locals=2, args\_size=2

0: getstatic #8 // Field java/lang/System.out:Ljava/io/PrintStream;

3: new #10 // class java/lang/StringBuilder

6: dup

7: invokespecial #11 // Method java/lang/StringBuilder."<init>":()V

10: aload\_0

11: invokevirtual #12 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;

14: aload\_1

15: invokevirtual #12 // Method java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;

18: invokevirtual #13 // Method java/lang/StringBuilder.toString:()Ljava/lang/String;

21: invokevirtual #9 // Method java/io/PrintStream.println:(Ljava/lang/String;)V

24: return

LineNumberTable:

line 16: 0

}

SourceFile: "LambdaTest.java"

InnerClasses:

public static final #74= #73 of #77; //Lookup=class java/lang/invoke/MethodHandles$Lookup of class java/lang/invoke/MethodHandles

BootstrapMethods:

0: #36 invokestatic java/lang/invoke/LambdaMetafactory.metafactory:(

Ljava/lang/invoke/MethodHandles$Lookup;

Ljava/lang/String;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodHandle;

Ljava/lang/invoke/MethodType;)

Ljava/lang/invoke/CallSite;

Method arguments:

#37 (Ljava/lang/String;)V

#38 invokestatic test/LambdaTest.lambda$main$0:(Ljava/lang/String;Ljava/lang/String;)V

#37 (Ljava/lang/String;)V

1: #36 invokestatic java/lang/invoke/LambdaMetafactory.metafactory:(

Ljava/lang/invoke/MethodHandles$Lookup;

Ljava/lang/String;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodHandle;

Ljava/lang/invoke/MethodType;)

Ljava/lang/invoke/CallSite;

Method arguments:

#37 (Ljava/lang/String;)V

#41 invokestatic test/LambdaTest.lambda$main$1:(Ljava/lang/String;)V

#37 (Ljava/lang/String;)V

这个 class 文件展示了三个主要部分：

* 常量池
* 构造方法和 main、print、lambda$main$0、lambda$main$1方法
* lambda表达式生成的内部类。

重点看下main方法的实现：

public static void main(java.lang.String[]);

descriptor: ([Ljava/lang/String;)V

flags: ACC\_PUBLIC, ACC\_STATIC

Code:

stack=2, locals=3, args\_size=1

// 将字符串常量"Chen Longfei"从常量池压栈到操作数栈

0: ldc #3 // String Chen Longfei

// 将栈顶引用型数值存入第二个本地变，即 String name = "Chen Longfei"

2: astore\_1

// 将字符串常量"hello,"从常量池压栈到操作数栈

3: ldc #4 // String hello,

// 将栈顶引用型数值存入第三个本地变量， 即 String prefix = "hello, "

5: astore\_2

//将第二个引用类型本地变量推送至栈顶，即 name

6: aload\_1

//将第三个引用类型本地变量推送至栈顶，即 prefix

7: aload\_2

//通过invokedynamic指令创建Print接口的实匿名内部类，实现 (t) -> System.out.println(prefix + t)

8: invokedynamic #5, 0 // InvokeDynamic #0:print:(Ljava/lang/String;)Ltest/Print;

//调用静态方法print

13: invokestatic #6 // Method print:(Ljava/lang/String;Ltest/Print;)V

//将第二个引用类型本地变量推送至栈顶，即 name

16: aload\_1

//通过invokedynamic指令创建Print接口的匿名内部类，实现 (t) -> System.out.println(t)

17: invokedynamic #7, 0 // InvokeDynamic #1:print:()Ltest/Print;

//调用静态方法print

22: invokestatic #6 // Method print:(Ljava/lang/String;Ltest/Print;)V

25: return

……

两个匿名内部类是通过BootstrapMethods方法创建的：

//匿名内部类

InnerClasses:

public static final #74= #73 of #77; //Lookup=class java/lang/invoke/MethodHandles$Lookup of class java/lang/invoke/MethodHandles

BootstrapMethods:

//调用静态工厂LambdaMetafactory.metafactory创建匿名内部类1。实现了 (t) -> System.out.println(prefix + t)

0: #36 invokestatic java/lang/invoke/LambdaMetafactory.metafactory:(

Ljava/lang/invoke/MethodHandles$Lookup;

Ljava/lang/String;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodHandle;

Ljava/lang/invoke/MethodType;)

Ljava/lang/invoke/CallSite;

Method arguments:

#37 (Ljava/lang/String;)V

//该类会调用静态方法LambdaTest.lambda$main$0

#38 invokestatic test/LambdaTest.lambda$main$0:(Ljava/lang/String;Ljava/lang/String;)V

#37 (Ljava/lang/String;)V

//调用静态工厂LambdaMetafactory.metafactory创建匿名内部类2，实现了 (t) -> System.out.println(t)

1: #36 invokestatic java/lang/invoke/LambdaMetafactory.metafactory:(

Ljava/lang/invoke/MethodHandles$Lookup;

Ljava/lang/String;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodType;

Ljava/lang/invoke/MethodHandle;

Ljava/lang/invoke/MethodType;)

Ljava/lang/invoke/CallSite;

Method arguments:

#37 (Ljava/lang/String;)V

//该类会调用静态方法LambdaTest.lambda$main$1

#41 invokestatic test/LambdaTest.lambda$main$1:(Ljava/lang/String;)V

#37 (Ljava/lang/String;)V

# 扩展

Class文件组成 虚拟机 调用方法的命令