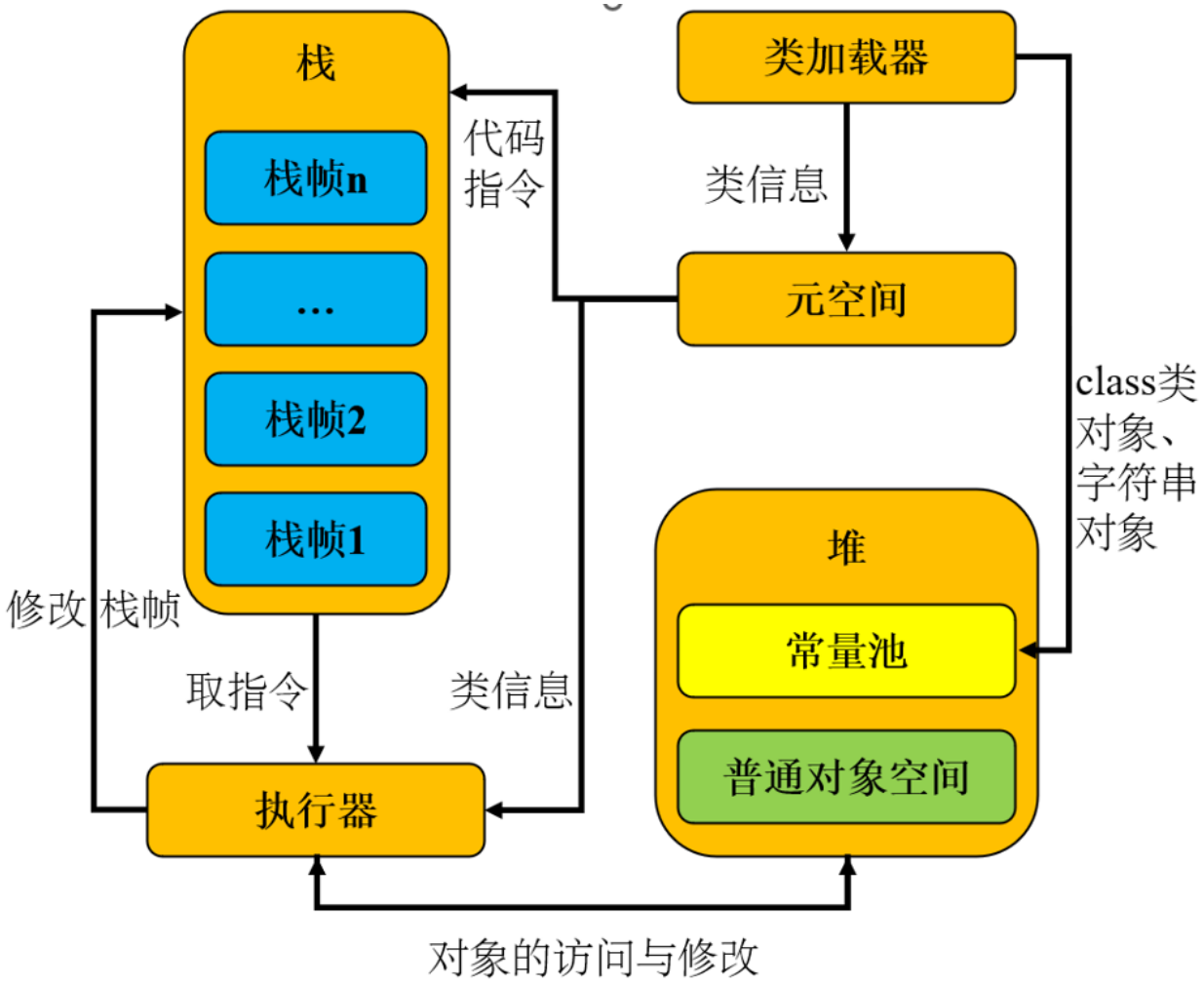
myLittleJVM的介绍

1 基本结构与说明

1.1 总体结构与主要的数据流动

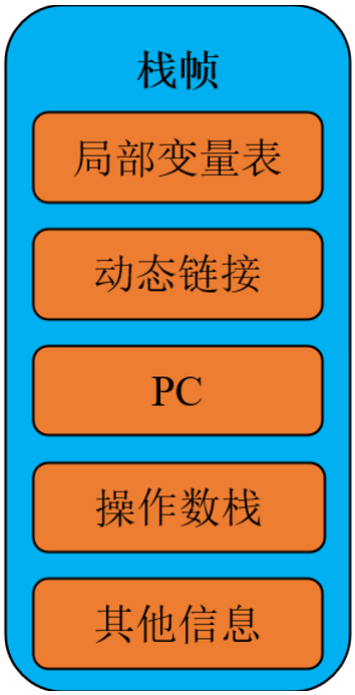
在总体结构上，myLittleJVM与常见的JVM是大约一致的，如下图。



myLittleJVM并没有将程序计数器（PC）单独列出，原因是myLittleJVM将PC放到了JVM栈的每一个栈帧里（如下图），这样做使方法的调用与返回在实现上更加容易。

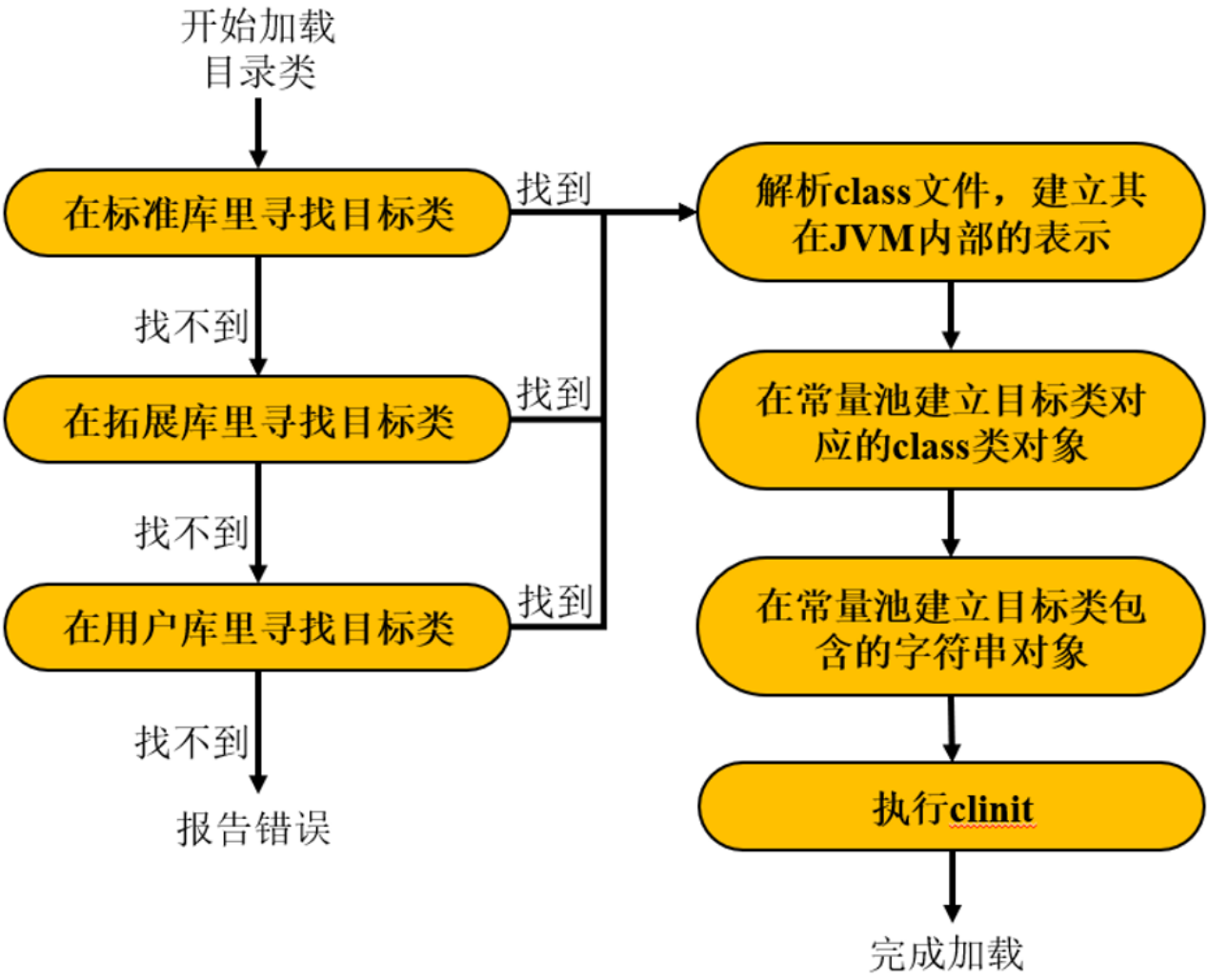
1.2 栈帧的结构

栈帧里的局部变量表和操作数栈和常见JVM相差不大，动态链接指向的是当前方法代码段的起始位置，其他信息里包含了本地方法的标识，用于表示当前方法是否为本地方法。



1.3 类加载的流程

在类加载器中，myLittleJVM没有采用双亲委派模型，而是采用了如下图的加载方式，这虽然会让加载器失去用户自定义的灵活性，但是这样做可以使得加载器的实现比较简单，同时加载速度快。下面加载方式是区分优先级的，这是为了保护核心类不被意外篡改（如无法自定义java.lang.String），这一点和双亲委派模型是一致的。



1.4 执行器

执行器实际上对应常见JVM的执行引擎。常见JVM的执行引擎是高度优化的，里面的结构十分复杂。为了实现的简单，myLittleJVM将执行器分为3部分，如下图。

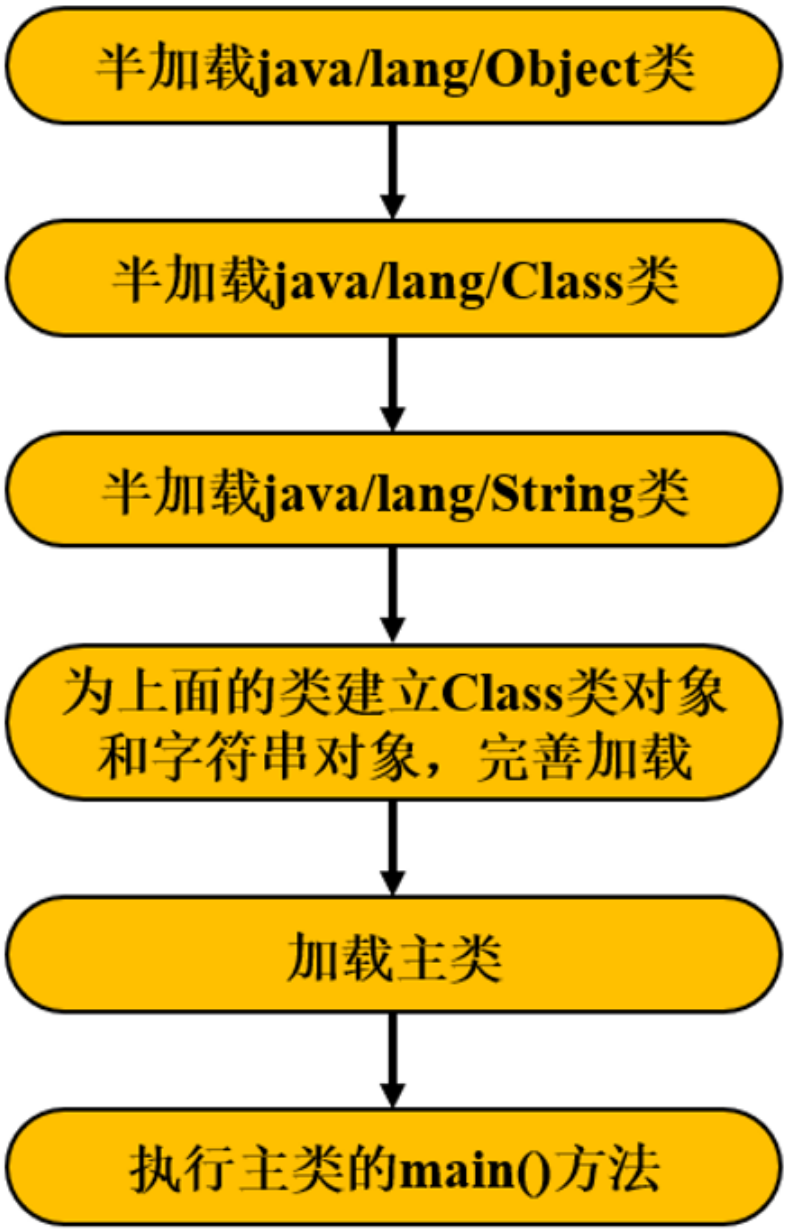
取值与分派：取出JVM栈顶一帧的指令，调用指令函数集合里面的对应函数完成指令动作。

指令的函数集合：一系列函数，与字节码指令一一对应。myLittleJVM实现了绝大部分指令，除了线程相关指令和invokeDynamic。

本地方法的函数集合：因为myLittleJVM与常见JVM（常见的JVM是工业级的，内部复杂）在现实上是不一致的，所以JNI难以统一。因此，myLittleJVM未去实现JNI，而是将运行中常见的本地方法直接实现，并存放到本地方法的函数集合中。在调用本地方法时，执行器会为其建立一个JVM栈帧，压入栈顶，复制参数到本地方法的局部变量表里，然后在集合里找方法并运行，本地方法将运行结果返回给执行器，执行器便销毁栈顶的本地方法的栈帧，将结果写入到调用者栈帧里面。因此，我们可以发现，myLittleJVM的本地方法的函数集合越丰富，它就可以运行越丰富的Java程序。

1.5 预加载

为了使主类的加载更加快速方便，myLittleJVM也像其他JVM一样，在启动时预加载一些重要的基本类，再加载主类并运行其main()方法。myLittleJVM会预先加载Object、Class、String类，流程如下。



2 支持的功能

Java语言的基本运算、控制结构和过程抽象

Java语言的面向对象

Java语言的字符串处理

Java语言集合框架

3 部分运行结果

测试样例使用的JDK是Java8u451。先编写代码，再进行编译产生字节码文件，最后交由myLittleJVM执行。部分样例会通过Java8u451的JVM运行，以验证myLittleJVM执行结果的正确性。下面是不同的测试。

3.1 快速排序

快速排序涉及到了Java语言的基本运算、控制结构和过程抽象。在myLittleJVM上执行Sort.class，结果正确，如下图。

```
C:\Users\YZQ\Desktop\as\myLittleJVM\cmake-build-debug\myLittleJVM.exe
输入主类名字:Sort
-----Sort.class运行输出-----
10
633 541 372 731 363 89 759 239 294 208
89 208 239 294 363 372 541 633 731 759
-----Sort.class正常退出-----

进程已结束，退出代码为 0
```

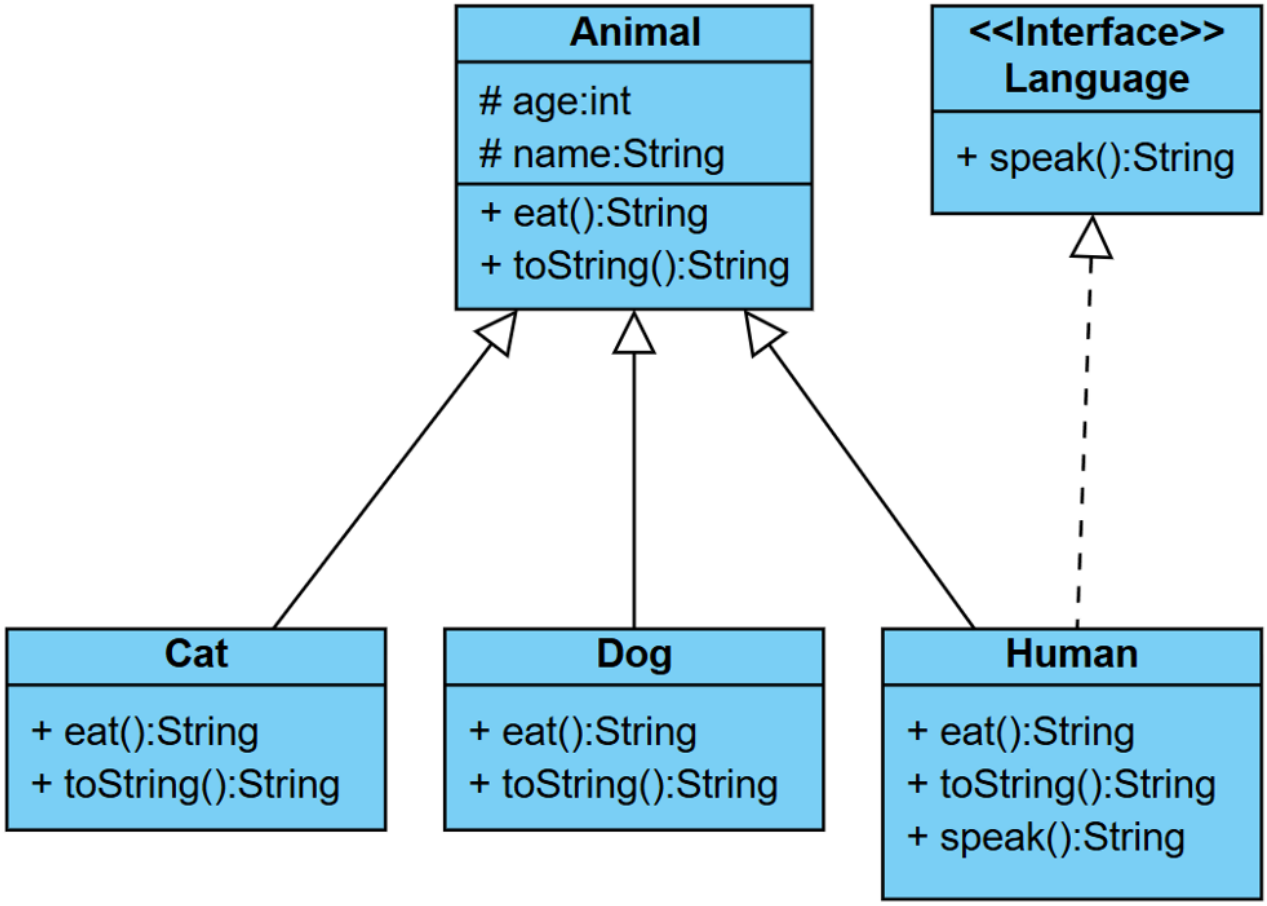
3.2 欧拉筛

欧拉筛涉及到了Java语言的基本运算和控制结构。在myLittleJVM上执行EulerSieve.class，结果正确，如下图。

```
C:\Users\YZQ\Desktop\as\myLittleJVM\cmake-build-debug\myLittleJVM.exe
输入主类名字:EulerSieve
-----EulerSieve.class运行输出-----
欧拉筛，请输入n:
29
2 3 5 7 11 13 17 19 23 29 -----EulerSieve.class正常退出-----

进程已结束，退出代码为 0
```

3.3 Zoo

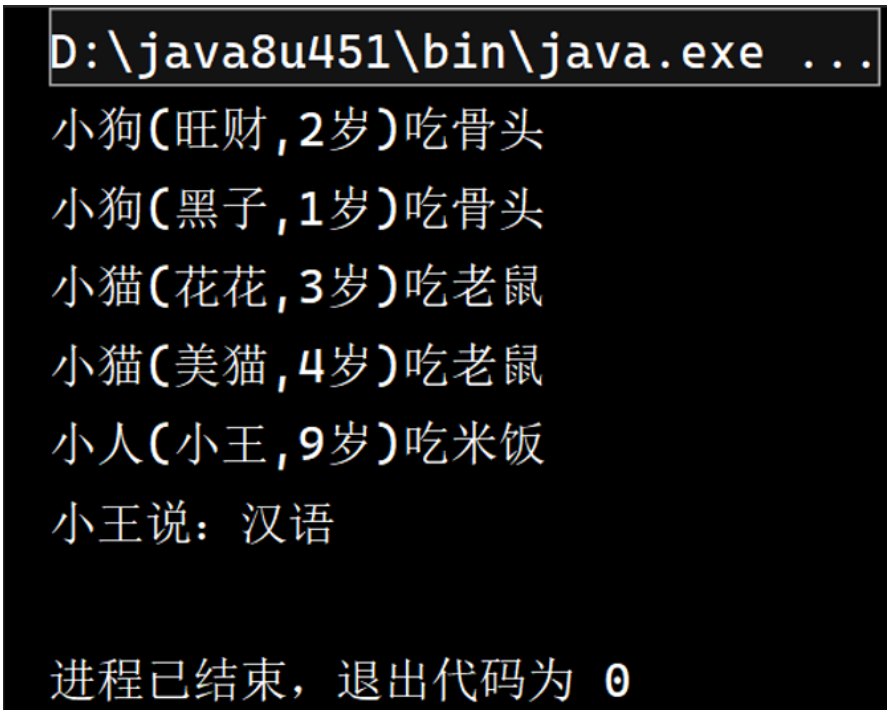
Zoo模拟了人、猫和狗吃东西，人讲一门语言的简单情景。Zoo情景的UML图如下。



Zoo情景涉及到了Java语言的面向对象特性，在myLittleJVM上执行Zoo.class，如下图。

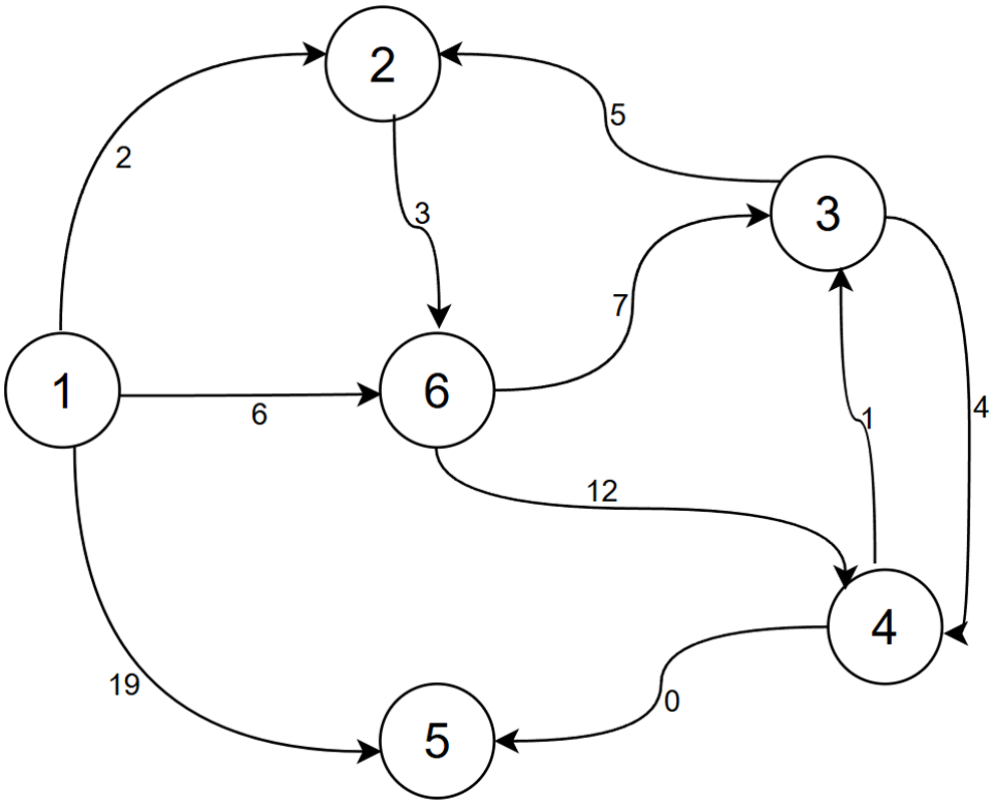
```
C:\Users\YZQ\Desktop\as\myLittleJVM\cmake-build-debug\myLittleJVM.exe
输入主类名字:Zoo
-----Zoo.class运行输出-----
小狗(旺财,2岁)吃骨头
小狗(黑子,1岁)吃骨头
小猫(花花,3岁)吃老鼠
小猫(美猫,4岁)吃老鼠
小人(小王,9岁)吃米饭
小王说：汉语
-----Zoo.class正常退出-----

进程已结束，退出代码为 0
```

为了验证myLittleJVM的执行正确性，在Java8的HotSpotVM上执行Zoo.class，如下图。myLittleJVM正确执行Zoo.class。



3.4 最短路径

编写一个堆优化的迪杰斯特拉最短路径算法，得到最短路径并打印沿途的权重。该算法涉及到了Java语言集合框架。输入的图如下。



在myLittleJVM上执行CollectionTest.class，如下图。

```
C:\Users\YZQ\Desktop\as\myLittleJVM\cmake-build-debug\myLittleJVM.exe
输入主类名字:CollectionTest
-----CollectionTest.class运行输出-----
6 10
1 2 2
1 6 6
1 5 19
2 6 3
3 2 5
3 4 4
4 3 1
4 5 0
6 3 7
6 4 12
1 5
edge(1->2) weight=2
edge(2->6) weight=3
edge(6->3) weight=7
edge(3->4) weight=4
edge(4->5) weight=0
-----CollectionTest.class正常退出-----

进程已结束，退出代码为 0
```

为了验证myLittleJVM的执行正确性，在Java8的HotSpotVM上执行CollectionTest.class，如下图。myLittleJVM正确执行CollectionTest.class。

```
D:\java8u451\bin\java.exe ...
已连接到地址为 ''127.0.0.1:60004'，传输: '套接字'' 的目标虚拟机
6 10
1 2 2
1 6 6
1 5 19
2 6 3
3 2 5
3 4 4
4 3 1
4 5 0
6 3 7
6 4 12
1 5
edge(1->2) weight=2
edge(2->6) weight=3
edge(6->3) weight=7
edge(3->4) weight=4
edge(4->5) weight=0
已与地址为 ''127.0.0.1:60004'，传输: '套接字'' 的目标虚拟机断开连接

进程已结束，退出代码为 0
```

4 改进方向

进一步完善和丰富本地方法的函数集合，以支持myLittleJVM的功能。myLittleJVM未建立真正与Java标准库匹配的IO机制，这是因为涉及到了大量的本地方法，笔者因时间原因未将其完整实现，而是使用自定义的myIO类来完成IO，这在测试代码中可以看见。myIO类在javaClassFile文件夹下。

添加垃圾回收功能。垃圾回收功能是JVM重要的功能，很遗憾，笔者因时间原因也未实现。不过笔者给出改进方向。我们可以让执行器的指令执行与垃圾回收器去互斥访问和修改堆空间里的普通对象空间，同时每一条指令的执行必须是原子性的。在垃圾回收时，可以使用根可达算法，根是栈帧里的局部变量表和操作数栈的引用元素。笔者在设计引用在JVM内部表示时，使用的是一个无符号32位整数，这会导致垃圾回收在寻找根时比较困难，尤其是在

操作数栈中。笔者建议为引用设置一个类，有必要时可以重新设计局部变量表、操作数栈等。

实现invokeDynamic指令。invokeDynamic指令是lambda的基础。invokeDynamic背后比较复杂，与Java标准库关联大。有兴趣和时间的伙伴可以研究一下。

实现即时编译。myLittleJVM是对指令解释执行，还没有将其编译成机器码运行。我们可以将热点的局部字节码指令段转化为机器码以加快执行速度，这涉及到了编译器的后端。

5 其他说明

myLittleJVM在Windows 11上使用CLion编写，语言版本为 C++ 11，编译环境为MinGW 11.0w64。执行的Java字节码由Java8u451的JDK产生。

项目中javaClassFile存放的是Java的标准类库，userClassFile存放用户的编译好的字节码文件，myLittleJVM从此处获取用户的字节码文件。

myLittleJVM在编写过程中，主要参考了 *The Java Virtual Machine Specification Java SE 8 Edition* 和《自己动手写Java虚拟机》。