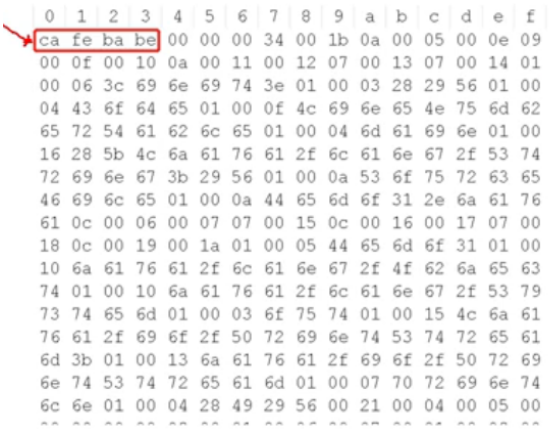
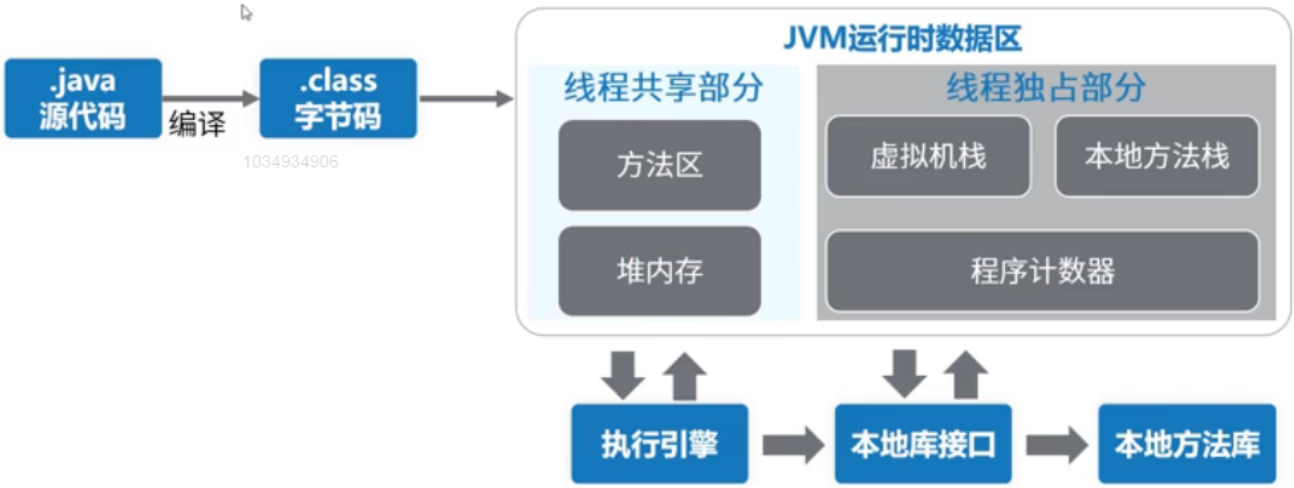
Java程序运行原理分析

class文件的内容

1. class文件包含JAVA程序执行的字节码；数据严格按照格式紧凑排列在class文件中的二进制流中，中间无任何分割符；文件开头有一个0xcafebabe特殊的一个标志。



JVM运行时数据区



如上图所示，JVM运行时数据区分为线程独享的虚拟机栈，本地方法栈，程序计数器，以及线程共享的方法区，堆内存。其中线程共享的区域，所有线程都能访问，随虚拟机或者GC而创建或者销毁。线程私有的区域，每个线程都会有它独立的空间，随线程生命周期而创建和销毁。

1. 方法区

   JVM用来存储加载的类信息，常量，静态变量，编译后的代码等数据。虚拟机规范中这是一个逻辑区划。具体实现根据不同虚拟机来实现。Oracle的HotSpot在java7中方法区放在永久代，java8放在元空间，并且通过GC机制对这个区域进行管理。

2. 堆内存

堆内存可以细分为：老年代，新生代(Eden，From Survivor, To Survivor)，JVM启动时创建，存 放对象的实例。垃圾回收器主要就是管理堆内存。如果对内存满了，就会出现OutOfMemoryError。

3. 虚拟机栈

   每个线程都在这个空间有一个私有的空间。线程栈由多个栈帧组成。一个线程会执行一个或多个方法，一个方法对应一个栈帧。栈帧中包含：局部变量表，操作数栈，动态链接，方法返回地址，附加信息等。栈的默认内存最大是1M，超过则会抛出StackOverflowError。

4. 本地方法栈

   和虚拟机栈类似，虚拟机栈是为虚拟机执行JAVA方法而准备的。本地方法栈是为虚拟机执行Native本地方法而准备的。虚拟机规范中没有具体的实现，由不同的虚拟机厂商去实现。HotSpot虚拟机中虚拟机和本地方法栈的实现是一样的。超出大小以后会抛出StackOverflowError。

5. 程序计数器

   记录当前线程执行字节码的位置，存储的是字节码指令的地址，如果执行Native方法，则计数器值为空。每个线程都在这个空间有一个私有的空间，占用内存空间很少。CPU同一时间，只会执行一条线程中的指令。JVM多线程会轮流切换并分配CPU执行时间的方式。线程切换后，需要通过程序计数器，来恢复正确的执行地址。

查看class文件的内容

1. 首先我们来看一个示例代码

```
package com.study.hc.thread.chapter1.thread;

public class Demo1 {

    public int x;

    public int sum(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        Demo1 demo1 = new Demo1();
        demo1.x = 3;
        int y = 2;
        int z = demo1.sum(demo1.x, y);
        System.out.println("person age is " + z);
    }
}
```

2. 我们首先使用javac Demo1.java命令，将Demo1.java编译成Demo1.class文件，然后用命令Javap Demo1.class > Demo1.txt得到解析后的可读性更好的class文件定义。

```
Classfile /C:/Users/zhu/Desktop/SeniorJava/Java/¿ÎÌÃ±Ê¼Ç/ÕýÊ½¿Î/¶àÏß³Ì²¢·¢±à³Ì/Java»ù´¡/×¨Ìâ»-µÚ»ØÂ/Demo1.class
  Last modified 2019-4-25; size 795 bytes
  MD5 checksum e4a628b971c32236aec9fb9b2672408d
```

```
  Compiled from "Demo1.java"
public class com.study.hc.thread.chapter1.thread.Demo1
  minor version: 0
  major version: 52
  flags: ACC_PUBLIC, ACC_SUPER
Constant pool:
   #1 = Methodref          #14.#27        // java/lang/Object."<init>":()V
   #2 = Class              #28            // com/study/hc/thread/chapter1/thread/Demo1
   #3 = Methodref          #2.#27         //
com/study/hc/thread/chapter1/thread/Demo1."<init>":()V
   #4 = Fieldref           #2.#29         //
com/study/hc/thread/chapter1/thread/Demo1.x:I
   #5 = Methodref          #2.#30         //
com/study/hc/thread/chapter1/thread/Demo1.sum:(II)I
   #6 = Fieldref           #31.#32        //
java/lang/System.out:Ljava/io/PrintStream;
   #7 = Class              #33            // java/lang/StringBuilder
   #8 = Methodref          #7.#27         // java/lang/StringBuilder."<init>":()V
   #9 = String             #34            // person age is
  #10 = Methodref          #7.#35         // java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
  #11 = Methodref          #7.#36         // java/lang/StringBuilder.append:
(I)Ljava/lang/StringBuilder;
  #12 = Methodref          #7.#37         // java/lang/StringBuilder.toString:
()Ljava/lang/String;
  #13 = Methodref          #38.#39        // java/io/PrintStream.println:
(Ljava/lang/String;)V
  #14 = Class              #40            // java/lang/Object
  #15 = Utf8               x
  #16 = Utf8               I
  #17 = Utf8               <init>
  #18 = Utf8               ()V
  #19 = Utf8               Code
  #20 = Utf8               LineNumberTable
  #21 = Utf8               sum
  #22 = Utf8               (II)I
  #23 = Utf8               main
  #24 = Utf8               ([Ljava/lang/String;)V
  #25 = Utf8               SourceFile
  #26 = Utf8               Demo1.java
  #27 = NameAndType        #17:#18        // "<init>":()V
  #28 = Utf8               com/study/hc/thread/chapter1/thread/Demo1
  #29 = NameAndType        #15:#16        // x:I
  #30 = NameAndType        #21:#22        // sum:(II)I
  #31 = Class              #41            // java/lang/System
  #32 = NameAndType        #42:#43        // out:Ljava/io/PrintStream;
  #33 = Utf8               java/lang/StringBuilder
  #34 = Utf8               person age is
  #35 = NameAndType        #44:#45        // append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
  #36 = NameAndType        #44:#46        // append:(I)Ljava/lang/StringBuilder;
  #37 = NameAndType        #47:#48        // toString:()Ljava/lang/String;
  #38 = Class              #49            // java/io/PrintStream
```

```
  #39 = NameAndType        #50:#51        // println:(Ljava/lang/String;)V
  #40 = Utf8               java/lang/Object
  #41 = Utf8               java/lang/System
  #42 = Utf8               out
  #43 = Utf8               Ljava/io/PrintStream;
  #44 = Utf8               append
  #45 = Utf8               (Ljava/lang/String;)Ljava/lang/StringBuilder;
  #46 = Utf8               (I)Ljava/lang/StringBuilder;
  #47 = Utf8               toString
  #48 = Utf8               ()Ljava/lang/String;
  #49 = Utf8               java/io/PrintStream
  #50 = Utf8               println
  #51 = Utf8               (Ljava/lang/String;)V
{
  public int x;
    descriptor: I
    flags: ACC_PUBLIC

  public com.study.hc.thread.chapter1.thread.Demo1();
    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 int sum(int, int);
    descriptor: (II)I
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=3, args_size=3
         0: iload_1 // 0x1b 从局部变量1中装载int类型值入栈
         1: iload_2 // 0x1c 从局部变量2中装载int类型值入栈
         2: iadd
         3: ireturn // 0xac 返回int类型值
      LineNumberTable:
        line 10: 0

  public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=3, locals=4, args_size=1
         0: new           #2                  // class
com/study/hc/thread/chapter1/thread/Demo1
         3: dup
         4: invokespecial #3                  // Method "<init>":()V
         7: astore_1
         8: aload_1
         9: iconst_3
```

```
         10: putfield      #4                  // Field x:I
         13: iconst_2
         14: istore_2
         15: aload_1
         16: aload_1
         17: getfield      #4                  // Field x:I
         20: iload_2
         21: invokevirtual #5                  // Method sum:(II)I
         24: istore_3
         25: getstatic     #6                  // Field
java/lang/System.out:Ljava/io/PrintStream;
         28: new           #7                  // class java/lang/StringBuilder
         31: dup
         32: invokespecial #8                  // Method java/lang/StringBuilder."
<init>":()V
         35: ldc           #9                  // String person age is
         37: invokevirtual #10                 // Method
java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
         40: iload_3
         41: invokevirtual #11                 // Method
java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
         44: invokevirtual #12                 // Method
java/lang/StringBuilder.toString:()Ljava/lang/String;
         47: invokevirtual #13                 // Method java/io/PrintStream.println:
(Ljava/lang/String;)V
         50: return
       LineNumberTable:
         line 14: 0
         line 15: 8
         line 16: 13
         line 17: 15
         line 18: 25
         line 19: 50
}
SourceFile: "Demo1.java"
```

3. class内容-版本号/访问控制

```
public class com.study.hc.thread.chapter1.thread.Demo1
  minor version: 0  // 次版本号
  major version: 52 // 主版本号
  flags: ACC_PUBLIC, ACC_SUPER
```

版本号规则：JDK5，6，7，8分别对应49，50，51，52

访问标志：

访问标志：

标志名称	标志值	含义
ACC_PUBLIC	0x0001	是否为public类型
ACC_FINAL	0x0010	是否被声明为final，只有类可设置
ACC_SUPER	0x0020	是否允许使用invokespecial字节码指令，JDK1.2之后编译出来的类的这个标志为true
ACC_INTERFACE	0x0200	标志这个是一个接口
ACC_ABSTRACT	0x0400	是否为abstract类型，对于接口或抽象类来说，此标志值为true，其他值为false
ACC_SYNTHETIC	0x1000	标志这个类并非由用户产生的
ACC_ANNOTATION	0x2000	标识这个一个注解
ACC_ENUM	0x4000	标识这是一个枚举

4. class内容-常量池

   类信息包含的静态常量，编译后就能确认。

```
Constant pool:
   #1 = Methodref          #14.#27        // java/lang/Object."<init>":()V
   #2 = Class              #28            // com/study/hc/thread/chapter1/thread/Demo1
   #3 = Methodref          #2.#27         //
com/study/hc/thread/chapter1/thread/Demo1."<init>":()V
   #4 = Fieldref           #2.#29         //
com/study/hc/thread/chapter1/thread/Demo1.x:I
   #5 = Methodref          #2.#30         //
com/study/hc/thread/chapter1/thread/Demo1.sum:(II)I
   #6 = Fieldref           #31.#32        //
java/lang/System.out:Ljava/io/PrintStream;
   #7 = Class              #33            // java/lang/StringBuilder
   #8 = Methodref          #7.#27         // java/lang/StringBuilder."<init>":()V
   #9 = String             #34            // person age is
  #10 = Methodref          #7.#35         // java/lang/StringBuilder.append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
  #11 = Methodref          #7.#36         // java/lang/StringBuilder.append:
(I)Ljava/lang/StringBuilder;
  #12 = Methodref          #7.#37         // java/lang/StringBuilder.toString:
()Ljava/lang/String;
  #13 = Methodref          #38.#39        // java/io/PrintStream.println:
(Ljava/lang/String;)V
  #14 = Class              #40            // java/lang/Object
  #15 = Utf8               x
  #16 = Utf8               I
  #17 = Utf8               <init>
  #18 = Utf8               ()V
  #19 = Utf8               Code
  #20 = Utf8               LineNumberTable
  #21 = Utf8               sum
  #22 = Utf8               (II)I
  #23 = Utf8               main
  #24 = Utf8               ([Ljava/lang/String;)V
  #25 = Utf8               SourceFile
  #26 = Utf8               Demo1.java
  #27 = NameAndType        #17:#18        // "<init>":()V
  #28 = Utf8               com/study/hc/thread/chapter1/thread/Demo1
  #29 = NameAndType        #15:#16        // x:I
```

```
  #30 = NameAndType        #21:#22        // sum:(II)I
  #31 = Class              #41            // java/lang/System
  #32 = NameAndType        #42:#43        // out:Ljava/io/PrintStream;
  #33 = Utf8               java/lang/StringBuilder
  #34 = Utf8               person age is
  #35 = NameAndType        #44:#45        // append:
(Ljava/lang/String;)Ljava/lang/StringBuilder;
  #36 = NameAndType        #44:#46        // append:(I)Ljava/lang/StringBuilder;
  #37 = NameAndType        #47:#48        // toString:()Ljava/lang/String;
  #38 = Class              #49            // java/io/PrintStream
  #39 = NameAndType        #50:#51        // println:(Ljava/lang/String;)V
  #40 = Utf8               java/lang/Object
  #41 = Utf8               java/lang/System
  #42 = Utf8               out
  #43 = Utf8               Ljava/io/PrintStream;
  #44 = Utf8               append
  #45 = Utf8               (Ljava/lang/String;)Ljava/lang/StringBuilder;
  #46 = Utf8               (I)Ljava/lang/StringBuilder;
  #47 = Utf8               toString
  #48 = Utf8               ()Ljava/lang/String;
  #49 = Utf8               java/io/PrintStream
  #50 = Utf8               println
  #51 = Utf8               (Ljava/lang/String;)V
```

CONSTANT_utf8_info	UTF-8编码的字符串
CONSTANT_Integer_info	整形字面量
CONSTANT_Float_info	浮点型字面量
CONSTANT_Long_info	长整型字面量
CONSTANT_Double_info	双精度浮点型字面量
CONSTANT_Class_info	类或接口的符号引用
CONSTANT_String_info	字符串类型字面量
CONSTANT_Fieldref_info	字段的符号引用
CONSTANT_Methodref_info	类中方法的符号引用
CONSTANT_InterfaceMethodref_info	接口中方法的符号引用
CONSTANT_NameAndType_info	字段或方法的符号引用
CONSTANT_MothodType_info	标志方法类型
CONSTANT_MethodHandle_info	表示方法句柄
CONSTANT_InvokeDynamic_info	表示一个动态方法调用点

5. class内容-构造方法

```
 public com.study.hc.thread.chapter1.thread.Demo1();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0 // 0x2a 从局部变量0中装载引用类型值入栈
         1: invokespecial #1                  // 0xb7 调用编译时绑定的方法 Method
java/lang/Object."<init>":()V
         4: return // 0xb1 void函数返回
      LineNumberTable: // 建立了字节码偏移量到源代码行号之间的对应关系
        line 6: 0
```

从代码示例中可以看到，我们并没有定义构造函数。由此可见，没有显示定义构造函数时，会有隐式的无参构造函数。

6. class内容-程序入口main方法(有自己查表得出的指令码解释)

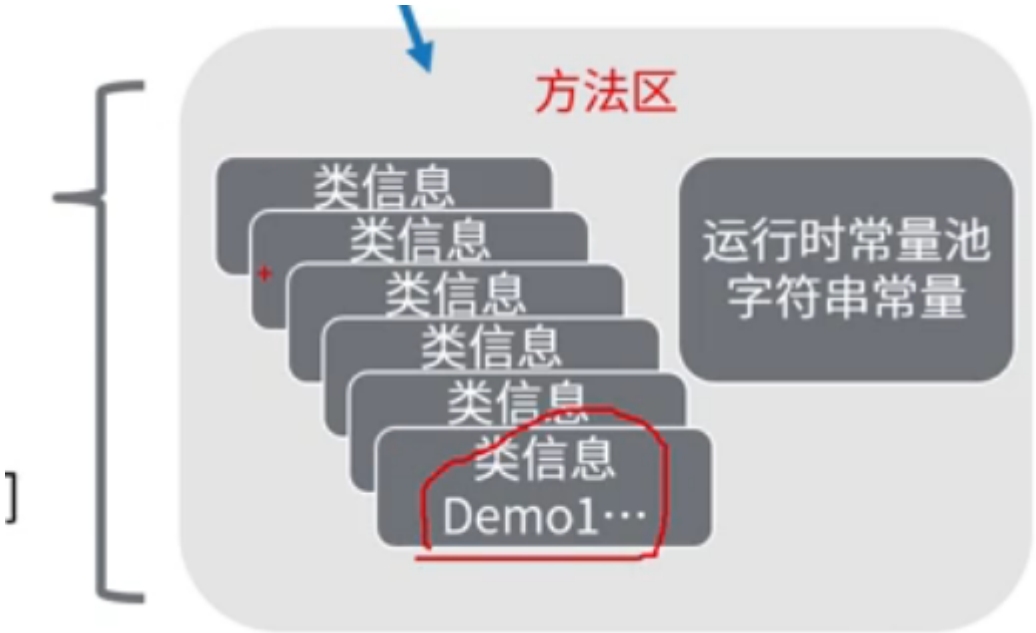
```
public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=3, locals=4, args_size=1
         0: new           #2                  // 0xbb 创建新的对象实例，在堆上为对象分配内存
空间，并将地址压入操作数栈 class com/study/hc/thread/chapter1/thread/Demo1
         3: dup // 0x59 复制栈顶一个字长的数据，将复制后的数据压栈，也就是说此时操作数栈上有连续相
同的两个对象地址，一个给实例的初始化方法用，一个给程序员使用，如果我们不用还是会生成dup指令，只不过在初
始化完成后再通过pop指令从栈顶移除
         4: invokespecial #3          // 0xb7 调用编译时绑定的方法
com/study/hc/thread/chapter1/thread/Demo1."<init>":()V
         7: astore_1 // 0x4c 将栈顶引用类型值保存到局部变量1中
         8: aload_1  // 0x2b 从局部变量1中装载引用类型的值入栈
         9: iconst_3 // 0x06 3(int)值入栈
        10: putfield      #4                  // 0xb5 给对象字段复制
com/study/hc/thread/chapter1/thread/Demo1.x:I Field x:I
        13: iconst_2 // 0x07 2(int)值入栈
        14: istore_2 // 0x3d 将栈顶int类型的值保存到局部变量2中
        15: aload_1  // 0x2b 从局部变量1中装载引用类型值入栈
        16: aload_1
        17: getfield      #4                  // 0xb4 获取对象字段的值
com/study/hc/thread/chapter1/thread/Demo1.x:I Field x:I
        20: iload_2 // 0x1c 从局部变量2中装载int类型的值入栈
        21: invokevirtual #5                  // 0xb6
com/study/hc/thread/chapter1/thread/Demo1.sum:(II)I Method sum:(II)I
        24: istore_3 // 0x3e 将栈顶int类型的值保存到局部变量3中
        25: getstatic     #6                  // 0xb2 获取静态字段的值 Field
java/lang/System.out:Ljava/io/PrintStream;
        28: new           #7                  // 0xbb 创建新的对象实例 class
java/lang/StringBuilder
        31: dup
        32: invokespecial #8                  // 0xb6 调用运行时绑定的方法 Method
java/lang/StringBuilder."<init>":()V
```

```
        35: ldc           #9                  // 0x12 常量池中的常量值入栈 String person
age is
        37: invokevirtual #10                 // 0xb6 调用运行时绑定的方法 Method
java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
        40: iload_3 // 0x21 从局部变量3中装载int类型的值入栈
        41: invokevirtual #11                 // 0xb6 调用运行时绑定的方法 Method
java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
        44: invokevirtual #12                 // 0xb6 调用运行时绑定的方法 Method
java/lang/StringBuilder.toString:()Ljava/lang/String;
        47: invokevirtual #13                 // 0xb6 调用运行时绑定的方法 Method
java/io/PrintStream.println:(Ljava/lang/String;)V
        50: return // 0xb1 void函数返回
```

其中，flags描述了方法的访问控制；locals，args_size，stack描述了方法的本地变量数量(单位为slot，为局部变量分配内存时使用的最小单位，为4个字节大小，locals的大小不一定等于所有局部变量所占slot之和，因为局部变量中的slot是可以重用的，包括实例方法隐含的this参数)，参数数量(实例方法包括this参数)和方法对应栈帧中操作数栈的深度。剩下的部分是javap翻译出来的操作符，JVM执行引擎去执行这些源码编译过后的指令码，class文件存储的是指令码。前面的数组，是偏移量(字节)，jvm根据这个区分不同的指令，操作符对应的指令可以查询JVM指令码表。

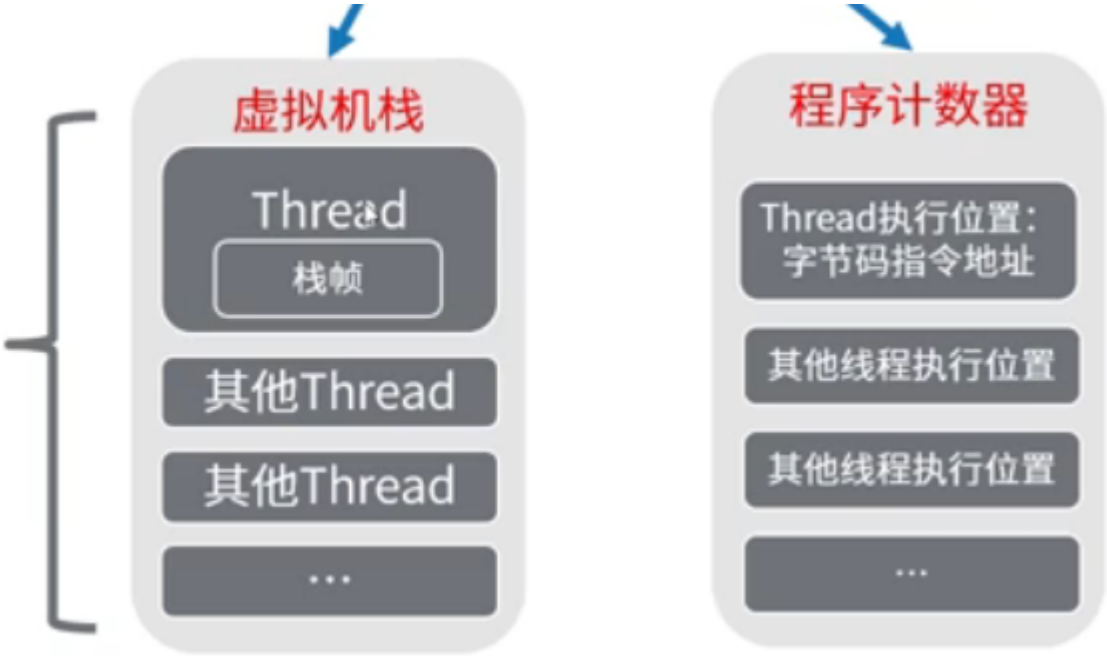
程序完整运行分析

1. 加载信息到方法区



2. JVM创建线程来执行代码

   在虚拟机栈，程序计数器，内存区域中创建线程独占的空间。



3. 程序运行分析

- 0：new指令创Demo1对象实例，为对象在堆上分配内存，并将地址压入操作数栈
- 3：dup指令将new出来的Demo1对象地址复制后再压栈
- 4：invokespecial指令从操作数栈中弹出一个Demo1的对象引用，调用该对象的默认构造函数
- 7：astore_1指令将操作数栈顶的Demo1的对象引用弹出，保存到局部变量1中，此时操作数栈为空
- 8：aload_1指令将保存在局部变量表1中的Demo1对象引用压栈
- 9：iconst_3指令将int型常数3压入操作数栈
- 10：putfield指令将栈顶的Demo1对象引用和常数3弹出，将Demo1对象的x字段赋值为3
- 13：iconst_2指令将int型常数2压入操作数栈
- 14：istore_2指令将栈顶的常数2保存到局部变量2中，即变量y中
- 15：aload_1指令将保存在局部变量表1中的Demo1对象引用压栈
- 16：aload_1指令将保存在局部变量表1中的Demo1对象引用压栈
- 17：getfield指令将操作数栈顶的Demo1对象引用弹出，获取Demo1对象的x字段，并将字段值压入操作数栈
- 20：iload_2指令从局部变量2将常数2压入操作数栈，此时操作数栈从栈顶到栈底为，常数2，对象x字段的值，以及Demo1对象引用
- 21：invokevirtual指令将三个操作数弹出栈，调用Demo1对象的sum方法
- 24：istore_3指令将调用sum方法的结果保存到局部变量3中
- 25：getstatic指令获取System类的静态字段out，将其引用压栈
- 28：new指令创建StringBuilder对象实例，为其在堆上分配内存，并将地址压入操作数栈
- 31：dup指令将new出来的StringBuilder对象地址复制后再压栈
- 32：invokespecial指令从操作数栈中弹出一个StringBuilder的对象引用，调用该对象的构造函数
- 35：idc指令将常量池中的“person age is”入栈
- 37：invokevirtual指令调用StringBuilder对象的append(String)方法，将返回的StringBuilder对象引用压栈
- 40：iload_3指令将局部变量3即z压入操作数栈
- 41：invokevirtual指令从操作数栈中弹出一个StringBuilder的对象引用，调用StringBuilder对象的append(int)方法，将返回的StringBuilder对象引用压栈
- 44：invokevirtual指令指令从操作数栈中弹出一个StringBuilder的对象引用，调用StringBuilder对象的toString()方法，将返回值String压栈
- 47：invokevirtual指令调用System的静态字段out的 println方法

- 50：return指令void函数返回

从上面的分析我们还可以看出，JVM底层使用StringBuilder类来处理相关的字符串操作的。