这篇文章我们将开始对象相关的字节码指令的介绍

0x01 new, <init> & <clinit>

在 Java 中 new 是一个关键字，在字节码中也有一个指令 new。当我们创建一个对象时，背后发生了哪些事情呢?

```
ScoreCalculator calculator = new
ScoreCalculator();
```

对应的字节码如下:

```
0: new           #2                  // class
ScoreCalculator
3: dup
4: invokespecial #3                  // Method
ScoreCalculator."<init>":()V

7: astore_1
```

一个对象创建的套路是这样的：new、dup、invokespecial，下次遇到同样的指令要形成条件反射。

为什么创建一个对象需要三条指令呢?
首先，我们需要清楚类的构造器函数是以<init>函数名出现的，被称为实例的初始化方法。调用 new 指令时，只是创建了一个类的实例，但是还没有调用构造器函数，使用 invokespecial 调用了 <init> 后才真正调用了构造器函数，正是因为需要调用这个函数才导致中间必须要有一个 dup 指令，不然调用完<init>函数以后，操作数栈为空，就再也找不回刚刚创建的对象了。

前面我们知道 <init> 会调用构造器函数，<clinit> 是类的静态初始化
比 <init> 调用得更早一些，<clinit> 不会直接被调用，它在下面这个四个指令触发调用：new，getstatic，putstatic or invokestatic。也就是说，初始化一个类实例、访问一个静态变量或者一个静态方法，类的静态初始化方法就会被触发。

看一个具体的例子

```
public class Initializer {
    static int a;
    static int b;
    static {
        a = 1;
        b = 2;
    }
}
```

部分字节码如下

```
static {};
     0: iconst_1
     1: putstatic     #2                  //
Field a:I
     4: iconst_2
     5: putstatic     #3                  //
Field b:I
     8: return
```

上面的 static {} 就对应我们刚说的 <clinit>

0x02 相关面试题分析

某东的一个面试题如下，类 A 和类 B 的关系如下

```
public class A {
    static {
        System.out.println("A init");
    }
    public A() {
        System.out.println("A Instance");
    }
}

public class B extends A {
    static {
        System.out.println("B init");
    }
    public B() {
        System.out.println("B Instance");
    }
}
```

问题 1： A a = new B(); 输出结果及正确的顺序?

要彻底搞清楚这个问题，需要弄清楚 B 构造器函数的字节码。

```
public B();
     0: aload_0
     1: invokespecial #1                  //
Method A."<init>":()V
     4: getstatic     #2                  //
Field java/lang/System.out:Ljava/io/PrintStream;
     7: ldc           #3                  //
String B Instance
     9: invokevirtual #4                  //
Method java/io/PrintStream.println:
(Ljava/lang/String;)V
    12: return
```

从 B 的构造器函数字节码可以看到它首先默默的帮忙调用了 A 的构造器函数
所以刚刚的过程是 new B() 的 <init> 触发了 B 的静态初始化 <clinit>，但这时父类 A 还没有进行静态初始化，会先进行 A 的静态初始化，然后执行 B 的构造器函数时，先调用了 A 的 <init> 构造器函数，最后执行 B 的构造器函数。

所以上述答案是：

```
A init
B init
A Instance
B Instance
```

> 问题 2：B[] arr = new B[10] 会输出什么？

这涉及到数组的初始化指令，对应字节码如下：

```
bipush 10
anewarray 'B'
astore 1
```

anewarray 接收栈顶的元素（数组的长度），新建一个数组引用。由此可见新建一个 B 的数组没有触发任何类或者实例的初始化操作。所以问题 2 的答案是什么也不会输出

> 问题3：如果把 B 的代码稍微改一下，新增一个静态不可变对象，调用System.out.println(B.HELLOWORD)会输出什么？

```
public class B extends A {
    public static final String HELLOWORD = "hello word";
    static{
        System.out.println("B init");
    }
    public B() {
        System.out.println("B Instance");
    }
}
public class InitOrderTest {
    public static void main(String[] args) {
        System.out.println(B.HELLOWORD);
    }
}
```

同样这里要回归到字节码和 JVM 本身
上，System.out.println(B.HELLOWORD)对应的字节码如下：

```
0: getstatic     #2                  // Field
java/lang/System.out:Ljava/io/PrintStream;
3: ldc           #4                  // String
hellow word
5: invokevirtual #5                  // Method
java/io/PrintStream.println:(Ljava/lang/String;)V

InitOrderTest 常量池信息如下:
#1 = Methodref          #7.#21         //
java/lang/Object."<init>":()V
#2 = Fieldref           #22.#23        //
java/lang/System.out:Ljava/io/PrintStream;
#3 = Class              #24            // B
#4 = String             #25            // hellow
word
```

可以看到同样没有触发任何 B 有关的初始化指令。虽然我们引用了 B 类的常量 HELLOWORD，但是这个常量在编译期间就被放到了 InitOrderTest 类的常量池中，不会与 B 发生任何关系
所以题目 3 的答案除了"hello world"以外什么也不会输出。

读者提问

有一个读者在群里提了一个问题：

> 为什么局部变量没有初始化，就不能使用。而一个对象的实例变量（无手动初始化）就可以用在后面的方法使用呢?

也就是下面的代码输出 0

```
public class TestLocal {
    int a;
    public static void main(String[] args) {
        TestLocal testLocal = new TestLocal();
        System.out.println(testLocal.a);
    }
}
```

而下面的代码编译出错，报error: variable b might not have been initialized

```
public class TestLocal {

    public void foo() {
        int b;
        System.out.println(b);
    }

    public static void main(String[] args) {
        TestLocal testLocal = new TestLocal();
        testLocal.foo();
    }
}
```

看起来是一个非常简单的问题，这背后的原理牵扯到 new 指令背后对象初始化的过程。以下面这个复杂一点的例子为例。

```
public class TestLocal {

    private int a;
    private static int b = 199;
    static {
        System.out.println("log from static
block");
    }
    public TestLocal() {
        System.out.println("log from constructor
block");
    }

    {
        System.out.println("log from init
block");
    }

    public static void main(String[] args) {
        TestLocal testLocal = new TestLocal();
    }
}

输出:
log from static block
log from init block
log from constructor block
```

如果去看源码的话，整个初始化过程简化如下（省略了若干步):

- 类加载校验：将类 TestLocal 加载到虚拟机
- 执行 static 代码块
- 为对象分配堆内存

- 对成员变量进行初始化（对象的实例字段在可以不赋初始值就直接使用，而局部变量中如果不赋值就直接使用，因为没有这一步操作，不赋值是属于未定义的状态，编译器会直接报错）
- 调用初始化代码块
- 调用构造器函数（可见构造器函数在初始化代码块之后执行）

弄清楚了这个流程，就很容易理解开始提出的问题了，简单来讲就是对象初始化的时候自动帮我们把未赋值的变量赋值为了初始值。

0x03 小结

这篇文章讲解了对象初始化相关的指令，一起来回顾一下要点：

- 第一，创建一个对象通常是 new、dup、<init>的 invokespecial 三条指令一起出现；
- 第二，类的静态初始化<clinit> 会在下面这个四个指令触发调用：new，getstatic，putstatic or invokestatic。

0x04 思考

最后，给你留一个道作业题，下面的代码会输出什么？原因是什么

```
class Father {
    private int i = test();
    private static int j = method();

    static {
        System.out.print("(1)");
    }

    Father() {
        System.out.print("(2)");
```

```
    }

    {
        System.out.print("(3)");
    }

    public int test() {
        System.out.print("(4)");
        return 1;
    }

    public static int method() {
        System.out.print("(5)");
        return 1;
    }
}

public class Son extends Father {
    private int i = test();
    private static int j = method();

    static {
        System.out.print("(6)");
    }

    Son() {
        System.out.print("(7)");
    }

    {
        System.out.print("(8)");
    }
```

```
    public int test() {
        System.out.print("(9)");
        return 1;
    }

    public static int method() {
        System.out.print("(10)");
        return 1;
    }

    public static void main(String[] args) {
        Son s1 = new Son();
        System.out.println();
        Son s2 = new Son();
    }
}
```

欢迎你在留言区留言，和我一起讨论。