链滴

JVM 运行时数据区域

作者：helly
原文链接：https://ld246.com/article/1510064996408
来源网站：链滴

一、JVM运行时数据区

1. 程序计数器（Program Counter Register）

Java虚拟机规范规定当线程在执行一个Java方法时，程序计数器记录的是当前正在执行的字节码指令当线程在执行的是本地方法时，该计数器的值是**未定义**的（undefined）。

该内存区域是唯一一个在Java虚拟机规范中么有规定任何OOM（内存溢出：OutOfMemoryError）况的区域。

对于Java虚拟机规范规定的“当前正在执行的字节码指令”，可以用指针实现，具体有两种方式，一是内存地址（bcp，bytecode pointer），一种是偏移量（bci，bytecode index）。

2. 方法区（Method Area）

Java虚拟机规范

存储虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。

JDK6及其以前的JDK版本

方法区的实现为永久代（Permanent Generation）。

存储：

类的信息，每个类都有一个运行时常量池（Run-time Constant Pool）；

通过字符串字面量创建的字符串对象、Java堆里的字符串对象调用了intern()方法后拷贝到方法区的符串对象，这些字符串对象就组成了字符串常量池（Interned String Pool）。

JDK7

方法区的实现还是永久代，但已经开始出现一些变化。

把StringTable引用的字符串对象移到Java堆中；

把静态变量从永久代（instanceKlass末尾）移动到了java.lang.Class对象的末尾；

把SymbolTable指向的对象从PermGen移动到了native memory。

JDK8

移除永久代，使用元空间。元空间并不在虚拟机内存区域中，而是使用本地内存。

Native Memory

Native Memory是相对于GC Heap的一个概念，不能处于GC Heap的区域就算Native Memory，JD 7以前的Java堆和永久带都属于GC Heap，JDK7及其以后的Java堆和元空间也属于GC Heap）

JIT编译器编译后的代码存储在**Native Memory**的**Code Cache**区域中（虽然Java虚拟机规范规定JI编译后的代码也应该属于方法区的一部分）。

字符串表（String Table），位于Native Memory只有一张，它里面存储的是字符串对象的引用，实的字符串对象在JDK6及其以前版本中是存储在永久代中，JDK7及其以后的版本是存储在Java堆中。

除了Float、Double外，其他包装类都有自己的常量池，Java虚拟机启动时就默认缓存了范围在[-128, 127]的对象，不在这个范围内的对象将永远不会被缓存到相应的常量池中，这些常量池是固定的，不字符串常量池是可以添加字符串对象的。

符号表（Symbol Table）

3. Java堆（Java Heap）

Java虚拟机规范规定Java堆存储所有对象。

数组也是对象，但由JVM为我们动态创建。数组继承Object类，多了个length属性

在JDK7以前，Hotspot虚拟机的永久代中存储了一些字符串对象。

对象内存布局

对象头（Header）

①Mark Word（32/64bit）

hashCode、GC分代年龄、锁状态标志、线程持有的锁、偏向线程ID、偏向时间戳。非固定结构，以在有限的空间尽可能存储多的信息。

②指向class对象的指针（32/64bit）

③长度（length）（如果是数组对象的话）（32/64bit）

实例数据（Instance Data）

包括自身还有从父类继承过来的，包括父类的private字段，只不过子类继承过来以后不能访问这个字。数据的存储顺序会受到虚拟机分配策略参数（FieldsAllocationStyle）和字段在 Java 源码中定义序的影响。

对齐填充（Padding）

不是必须，保证8字节对齐。

OOP/Klass二分模型

OOP（Ordinary Object Pointer）：存放实例信息；

Klass：存放元数据。

instanceKlass

instanceOopDesc表示Java对象，arrayOopDesc表示Java数组，klassOopDesc表示Java类。

4. Java虚拟机栈（Java Virtual Machine Stacks）

Java虚拟机栈由**栈帧**（Stack Frame）组成。每个栈帧又包括操作数栈、局部变量表、方法返回地址动态链接。操作数栈和局部变量表的大小在编译的时候就确定了，并且写入了方法表的Code属性中。

局部变量表（Local Variables）

我们在调用实例方法的时候，局部变量表的第0位是一个指向当前方法所属对象的引用，这个引用也是我们说的this关键字，它是作为方法的隐藏参数传进来的，所以我们可以在实例方法里通过它访问前对象；如果我们调用的是静态方法，第0位就不是this了，从第0位开始就直接开始放数据了。

局部变量表的单位是变量槽（Slot）。一个Slot可以存放一个32位以内的数据类型：boolean、byte char、short、int、float、reference和returnAddresss。对于64位的数据类型（long和double）虚拟机会以高位在前的方式为其分配两个连续的Slot空间。

局部变量表中的Slot是可重用的。

操作数栈（Operand Stack）

Java语言中，实例构造器只能在new表达式（或别的构造器）中被调用。

new表达式作为一个整体保证了对象的**创建**与**初始化**是打包在一起进行的，不能分开进行；但实例构器只负责对象初始化的部分，“创建对象”的部分是由new表达式本身保证的。

new一个对象过程：

1. new这个字节码指令做了两件事，一个是创建了一个对象，但还未初始化，也就是一个空对象，第件事就是把指向这个对象的引用压入操作数栈；

2. 执行dup指令，dup指令将栈顶元素，也就是指向刚刚创建好的那个空对象的引用取出来，然后复两份，一份保存到局部变量表里，因为构造器没有返回参数，待会构造器执行完了不会有引用返回回。另一份作为隐藏参数传给构造器，所以在构造器里可以使用this关键字。

3. 执行invokespecial指令，调用构造器，将这个对象进行初始化。

4. 构造器执行完毕后方法返回，原先的方法执行return指令，把原来记录在局部变量表的引用返回。

动态链接（Current Class Constant Pool Reference）

每个栈帧都包含一个指向运行时常量池中该栈帧所属方法的引用，持有这个引用是为了支持方法调用程中的动态连接。

方法返回地址（Return Value）

当一个方法执行完毕之后，要返回之前调用它的地方，因此在栈帧中必须保存一个方法返回地址，用恢复上一个方法的操作数栈和局部变量表。

5. 本地方法栈（Native Method Stacks）

Java虚拟机规范规定该区域是供本地方法执行用的。