21 | 磨刀不误砍柴工：欲知JVM调优先了解JVM内存模型

2019-07-09 刘超

你好，我是刘超。

从今天开始，我将和你一起探讨Java虚拟机（JVM）的性能调优。JVM算是面试中的高频问题了，通常情况下总会有人问到：请你讲解下JVM的内存模型，JVM的性能调优做过吗？

为什么JVM在Java中如此重要？

首先你应该知道，运行一个Java应用程序，我们必须要先安装JDK或者JRE包。这是因为Java应用在编译后会变成字节码，然后通过字节码运行在JVM中，而JVM是JRE的核心组成部分。

JVM不仅承担了Java字节码的分析（JIT compiler）和执行（Runtime），同时也内置了自动内存分配管理机制。这个机制可以大大降低手动分配回收机制可能带来的内存泄露和内存溢出风险，使Java开发人员不需要关注每个对象的内存分配以及回收，从而更专注于业务本身。

从了解内存模型开始

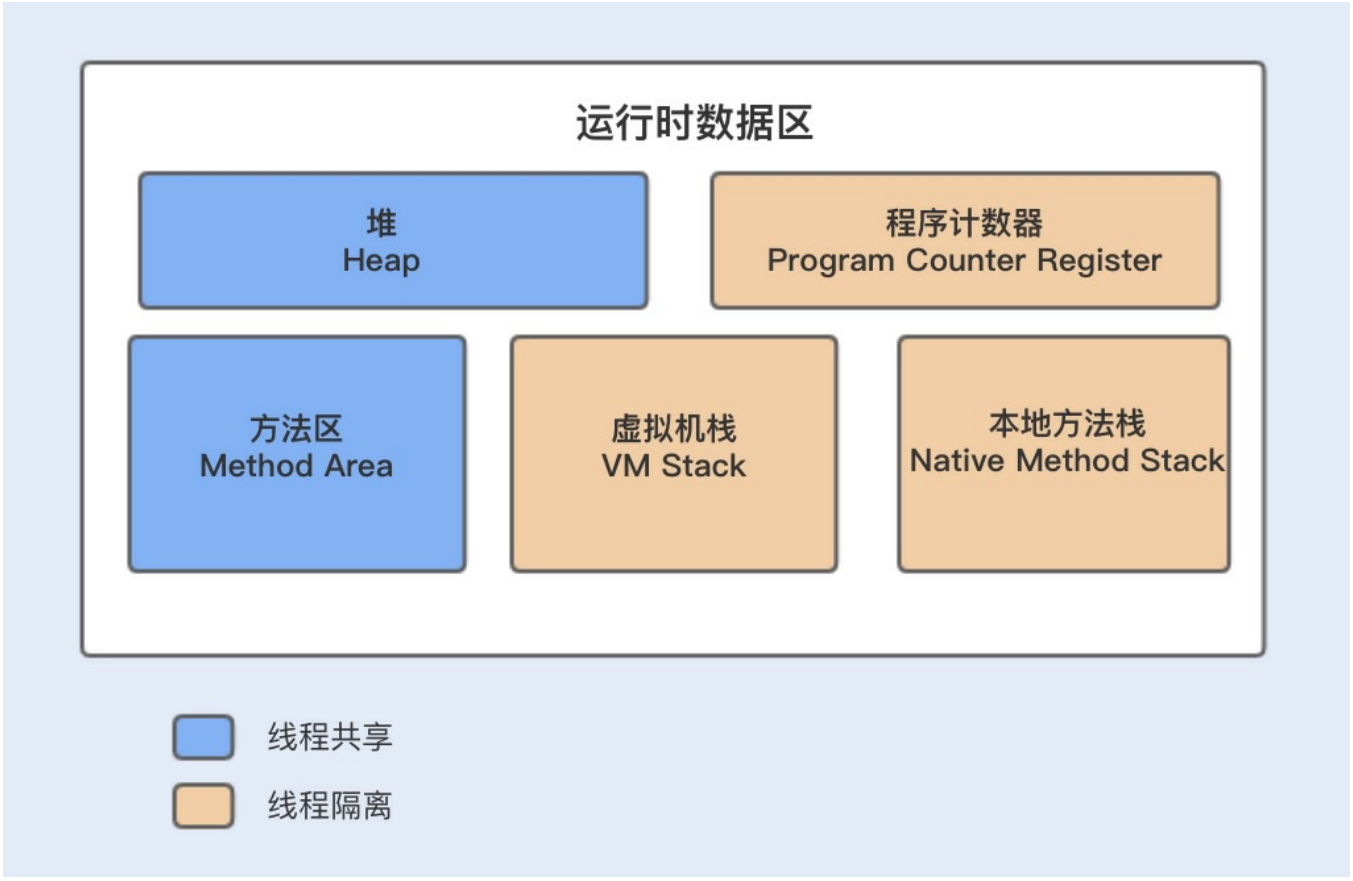
JVM自动内存分配管理机制的好处很多，但实则是把双刃剑。这个机制在提升Java开发效率的同时，也容易使Java开发人员过度依赖于自动化，弱化对内存的管理能力，这样系统就很容易发生JVM的堆内存异常，垃圾回收（GC）的方式不合适以及GC次数过于频繁等问题，这些都将直接影响到应用服务的性能。

因此，要进行JVM层面的调优，就需要深入了解JVM内存分配和回收原理，这样在遇到问题时，我们才能通过日志分析快速地定位问题；也能在系统遇到性能瓶颈时，通过分析JVM调优来优化

系统性能。这也是整个模块四的重点内容，今天我们就从JVM的内存模型学起，为后续的学习打下一个坚实的基础。

JVM内存模型的具体设计

我们先通过一张JVM内存模型图，来熟悉下其具体设计。在Java中，JVM内存模型主要分为堆、程序计数器、方法区、虚拟机栈和本地方法栈。

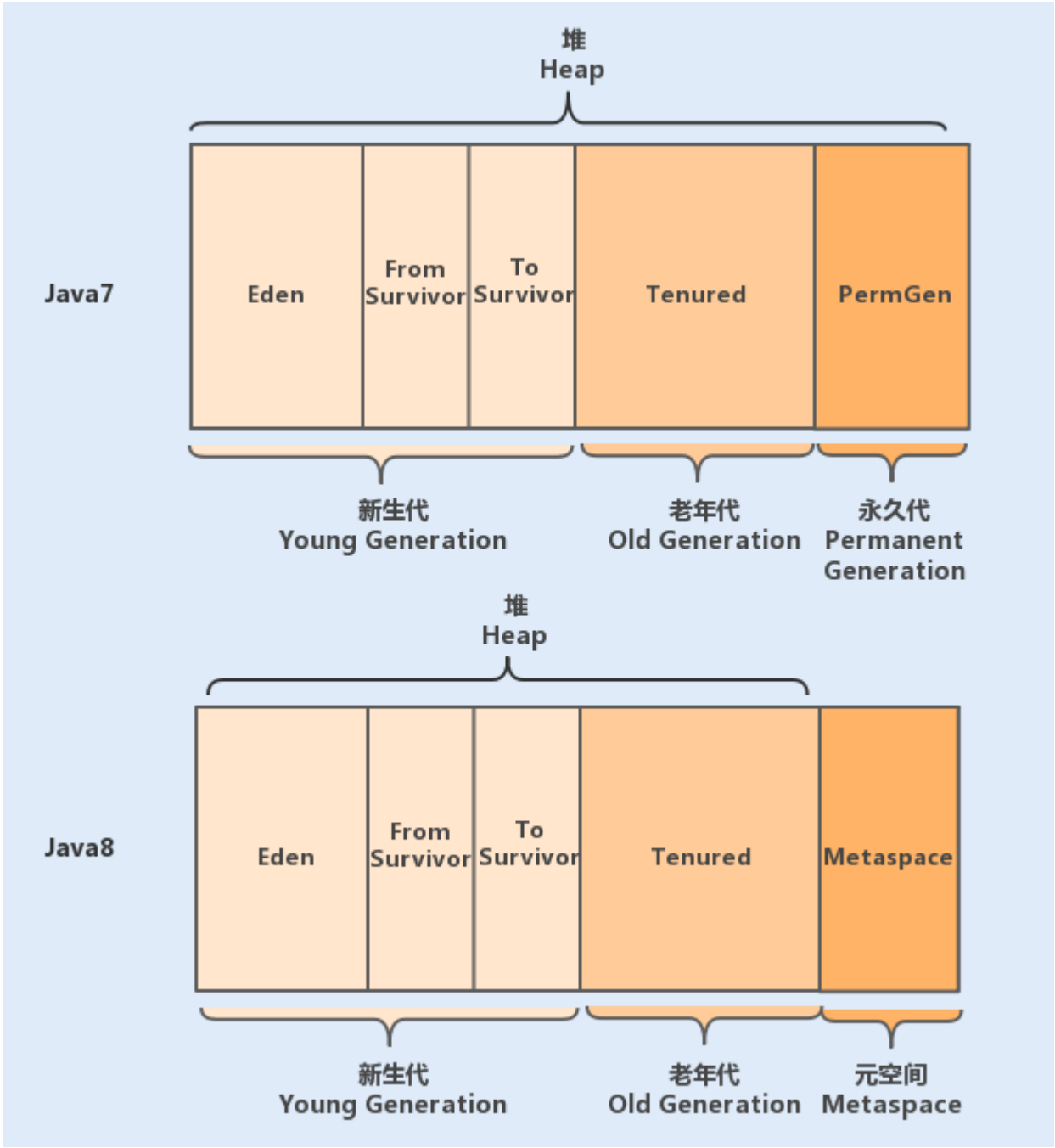


JVM的5个分区具体是怎么实现的呢？我们一一分析。

1. 堆（Heap）

堆是JVM内存中最大的一块内存空间，该内存被所有线程共享，几乎所有对象和数组都被分配到了堆内存中。堆被划分为新生代和老年代，新生代又被进一步划分为Eden和Survivor区，最后Survivor由From Survivor和To Survivor组成。

在Java6版本中，永久代在非堆内存区；到了Java7版本，永久代的静态变量和运行时常量池被合并到了堆中；而到了Java8，永久代被元空间取代了。 结构如下图所示：



2. 程序计数器（Program Counter Register）

程序计数器是一块很小的内存空间，主要用来记录各个线程执行的字节码的地址，例如，分支、循环、跳转、异常、线程恢复等都依赖于计数器。

由于Java是多线程语言，当执行的线程数量超过CPU核数时，线程之间会根据时间片轮询争夺CPU资源。如果一个线程的时间片用完了，或者是其它原因导致这个线程的CPU资源被提前抢夺，那么这个退出的线程就需要单独的一个程序计数器，来记录下一条运行的指令。

3. 方法区（Method Area）

很多开发者都习惯将方法区称为“永久代”，其实这两者并不是等价的。

HotSpot虚拟机使用永久代来实现方法区，但在其它虚拟机中，例如，Oracle的JRockit、IBM的J9就不存在永久代一说。因此，方法区只是JVM中规范的一部分，可以说，在HotSpot虚拟机

中，设计人员使用了永久代来实现了JVM规范的方法区。

方法区主要是用来存放已被虚拟机加载的类相关信息，包括类信息、运行时常量池、字符串常量池。类信息又包括了类的版本、字段、方法、接口和父类等信息。

JVM在执行某个类的时候，必须经过加载、连接、初始化，而连接又包括验证、准备、解析三个阶段。在加载类的时候，JVM会先加载class文件，而在class文件中除了有类的版本、字段、方法和接口等描述信息外，还有一项信息是常量池(Constant Pool Table)，用于存放编译期间生成的各种字面量和符号引用。

字面量包括字符串（String a="b"）、基本类型的常量（final修饰的变量），符号引用则包括类和方法的全限定名（例如String这个类，它的全限定名就是Java/lang/String）、字段的名称和描述符以及方法的名称和描述符。

而当类加载到内存中后，JVM就会将class文件常量池中的内容存放到运行时的常量池中；在解析阶段，JVM会把符号引用替换为直接引用（对象的索引值）。

例如，类中的一个字符串常量在class文件中时，存放在class文件常量池中的；在JVM加载完类之后，JVM会将这个字符串常量放到运行时常量池中，并在解析阶段，指定该字符串对象的索引值。运行时常量池是全局共享的，多个类共用一个运行时常量池，class文件中常量池多个相同的字符串在运行时常量池只会存在一份。

方法区与堆空间类似，也是一个共享内存区，所以方法区是线程共享的。假如两个线程都试图访问方法区中的同一个类信息，而这个类还没有装入JVM，那么此时就只允许一个线程去加载它，另一个线程必须等待。

在HotSpot虚拟机、Java7版本中已经将永久代的静态变量和运行时常量池转移到了堆中，其余部分则存储在JVM的非堆内存中，而Java8版本已经将方法区中实现的永久代去掉了，并用元空间（class metadata）代替了之前的永久代，并且元空间的存储位置是本地内存。之前永久代的类的元数据存储在了元空间，永久代的静态变量（class static variables）以及运行时常量池（runtime constant pool）则跟Java7一样，转移到了堆中。

那你可能又有疑问了，Java8为什么使用元空间替代永久代，这样做有什么好处呢？

官方给出的解释是：

- 移除永久代是为了融合 HotSpot JVM 与 JRockit VM 而做出的努力，因为JRockit没有永久代，所以不需要配置永久代。
- 永久代内存经常不够用或发生内存溢出，爆出异常java.lang.OutOfMemoryError: PermGen。这是因为在JDK1.7版本中，指定的PermGen区大小为8M，由于PermGen中类的元数据信息在每次FullGC的时候都可能被收集，回收率都偏低，成绩很难令人满意；还有，为PermGen

分配多大的空间很难确定，PermSize的大小依赖于很多因素，比如，JVM加载的class总数、常量池的大小和方法的大小等。

4.虚拟机栈（VM stack）

Java虚拟机栈是线程私有的内存空间，它和Java线程一起创建。当创建一个线程时，会在虚拟机栈中申请一个线程栈，用来保存方法的局部变量、操作数栈、动态链接方法和返回地址等信息，并参与方法的调用和返回。每一个方法的调用都伴随着栈帧的入栈操作，方法的返回则是栈帧的出栈操作。

5.本地方法栈（Native Method Stack）

本地方法栈跟Java虚拟机栈的功能类似，Java虚拟机栈用于管理Java函数的调用，而本地方法栈则用于管理本地方法的调用。但本地方法并不是用Java实现的，而是由C语言实现的。

JVM的运行原理

看到这里，相信你对JVM内存模型已经有个充分的了解了。接下来，我们通过一个案例来了解下代码和对象是如何分配存储的，Java代码又是如何在JVM中运行的。

```
public class JVMCase {

 // 常量
 public final static String MAN_SEX_TYPE = "man";

 // 静态变量
 public static String WOMAN_SEX_TYPE = "woman";

 public static void main(String[] args) {

  Student stu = new Student();
  stu.setName("nick");
  stu.setSexType(MAN_SEX_TYPE);
  stu.setAge(20);

  JVMCase jvmcase = new JVMCase();

  // 调用静态方法
  print(stu);
  // 调用非静态方法
  jvmcase.sayHello(stu);
```

```
 }


 // 常规静态方法
 public static void print(Student stu) {
  System.out.println("name: " + stu.getName() + "; sex:" + stu.getSexType() + "; age:" + stu.getAge());
 }


 // 非静态方法
 public void sayHello(Student stu) {
  System.out.println(stu.getName() + "say: hello");
 }
}

class Student{
 String name;
 String sexType;
 int age;

 public String getName() {
  return name;
 }
 public void setName(String name) {
  this.name = name;
 }

 public String getSexType() {
  return sexType;
 }
 public void setSexType(String sexType) {
  this.sexType = sexType;
 }
 public int getAge() {
  return age;
 }
 public void setAge(int age) {
  this.age = age;
```

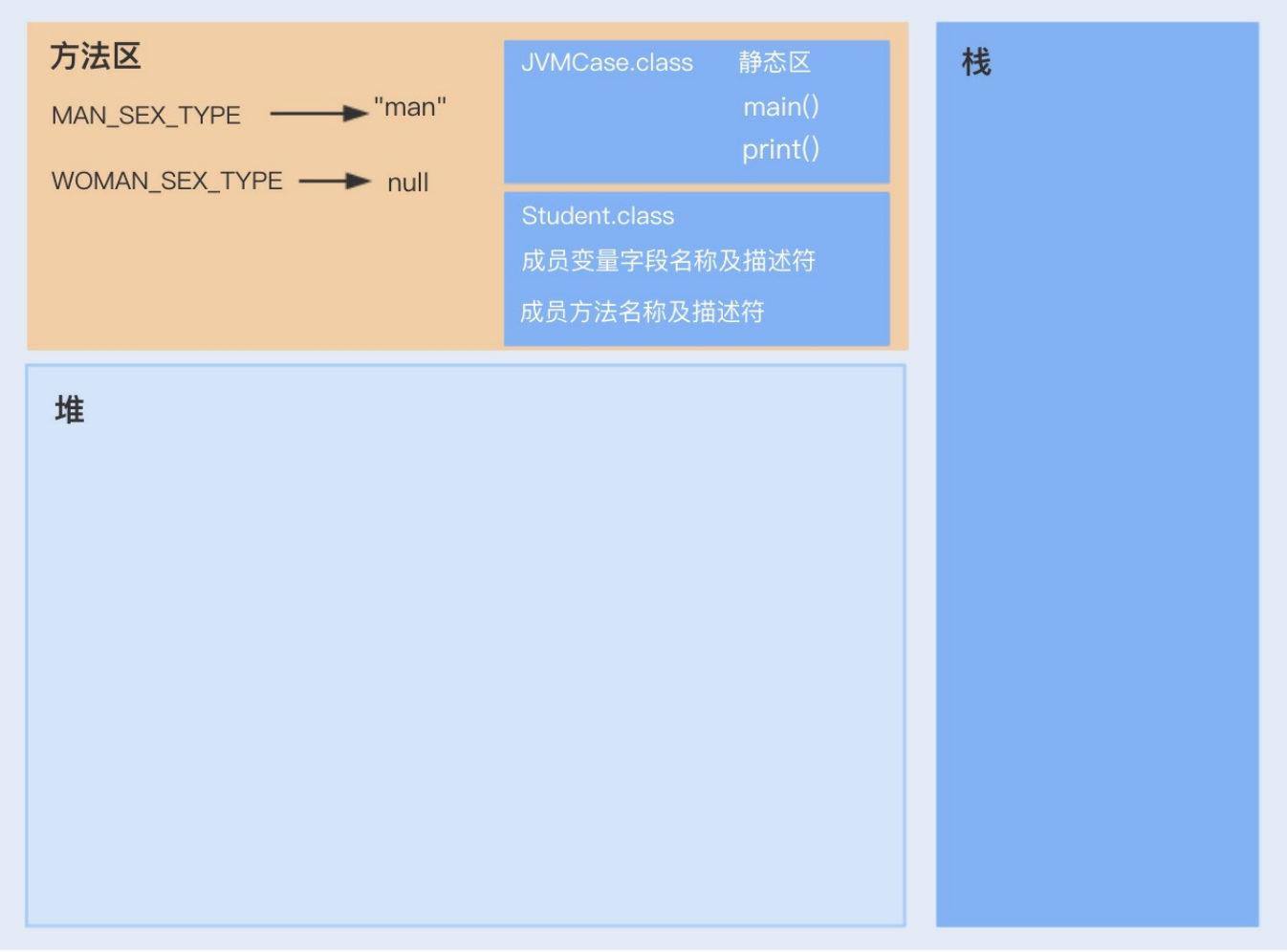
```
    this.age = age;
  }
}
```

当我们通过Java运行以上代码时，JVM的整个处理过程如下：

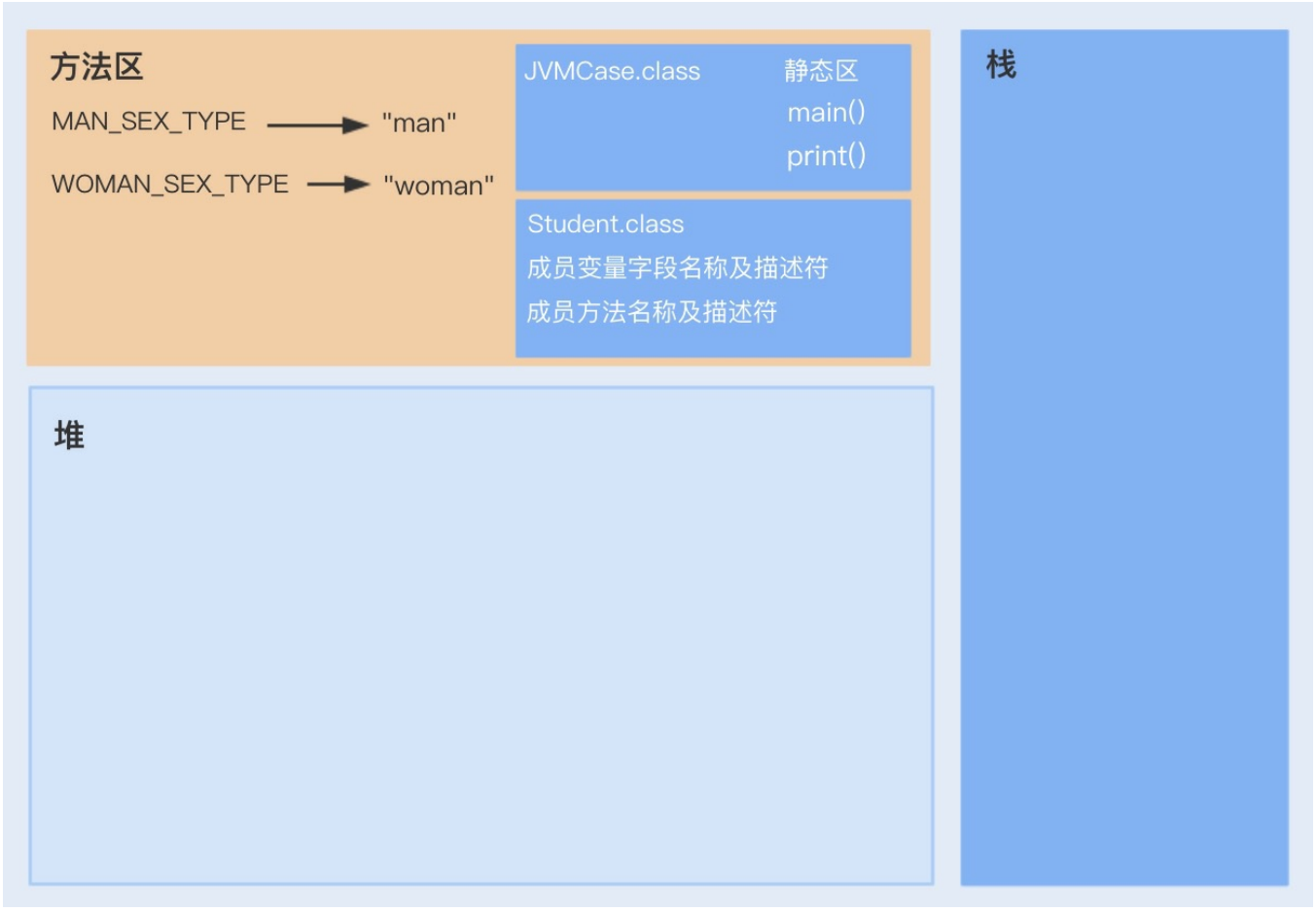
1.JVM向操作系统申请内存，JVM第一步就是通过配置参数或者默认配置参数向操作系统申请内存空间，根据内存大小找到具体的内存分配表，然后把内存段的起始地址和终止地址分配给JVM，接下来JVM就进行内部分配。

2.JVM获得内存空间后，会根据配置参数分配堆、栈以及方法区的内存大小。

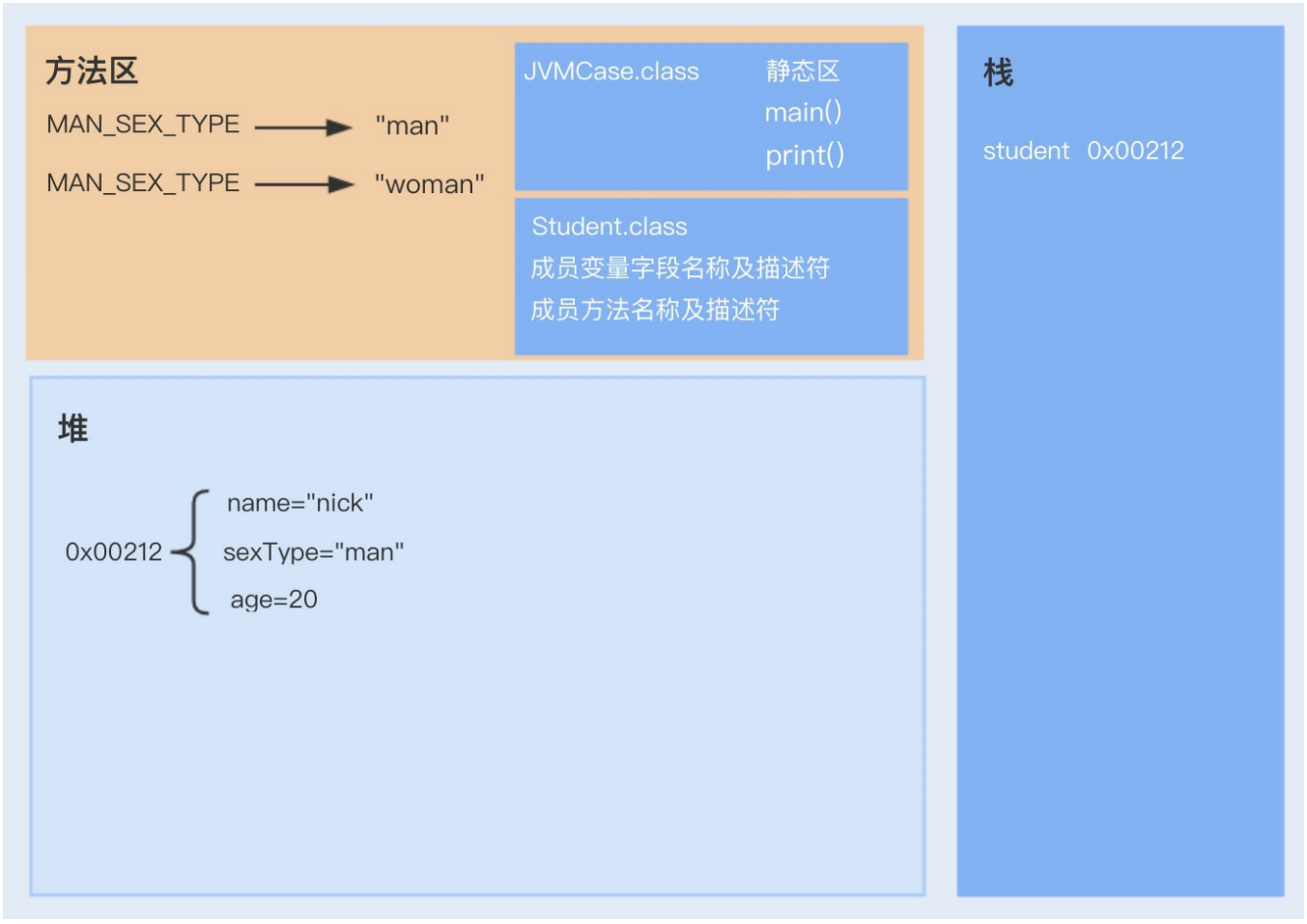
3.class文件加载、验证、准备以及解析，其中准备阶段会为类的静态变量分配内存，初始化为系统的初始值（这部分我在第21讲还会详细介绍）。



4.完成上一个步骤后，将会进行最后一个初始化阶段。在这个阶段中，JVM首先会执行构造器<clinit>方法，编译器会在.java 文件被编译成.class 文件时，收集所有类的初始化代码，包括静态变量赋值语句、静态代码块、静态方法，收集在一起成为 <clinit>() 方法。



5.执行方法。启动main线程，执行main方法，开始执行第一行代码。此时堆内存中会创建一个student对象，对象引用student就存放在栈中。



6.此时再次创建一个JVMCase对象，调用sayHello非静态方法，sayHello方法属于对象JVMCase，此时sayHello方法入栈，并通过栈中的student引用调用堆中的Student对象；之后，调用静态方法print，print静态方法属于JVMCase类，是从静态方法中获取，之后放入到栈中，也是通过student引用调用堆中的student对象。

了解完实际代码在JVM中分配的内存空间以及运行原理，相信你会更加清楚内存模型中各个区域的职责分工。

总结

这讲我们主要深入学习了最基础的内存模型设计，了解其各个分区的作用及实现原理。

如今，JVM在很大程度上减轻了Java开发人员投入到对象生命周期的管理精力。在使用对象的时候，JVM会自动分配内存给对象，在不使用的时候，垃圾回收器会自动回收对象，释放占用的内存。

但在某些情况下，正常的生命周期不是最优的选择，有些对象按照JVM默认的方式，创建成本会很高。比如，我在第03讲讲到的String对象，在特定的场景使用String.intern可以很大程度地节约内存成本。我们可以使用不同的引用类型，改变一个对象的正常生命周期，从而提高JVM的回收效率，这也是JVM性能调优的一种方式。

思考题

这讲我只提到了堆内存中对象分配内存空间的过程，那如果有一个类中定义了String a="b"和String c = new String(“b”)，请问这两个对象会分别创建在JVM内存模型中的哪块区域呢？

期待在留言区看到你的答案。也欢迎你点击“请朋友读”，把今天的内容分享给身边的朋友，邀请他一起讨论。

精选留言

张学磊 19

String a="b"可能创建一个对象或者不创建对象,如果"b"这个字符串在常量池里不存在会在常量池创建一个String对象"b",如果已经存在则a直接reference to这个常量池里的对象;
String c= new String("b")至少创建一个对象,也可能两个,因为用到new关键字,会在堆内在创建一个的String对象,它的值是"b"。同时,如果"b"这个字符串在常量池里不存在,会在常量池创建这个一个String对象"b"。

2019-07-09

作者回复

对的

2019-07-10

Xiao 11

老师，这儿其实应该说JVM内存结构更合适！JVM内存模型是一种规范，和JVM内存结构不是一个概念。其次，元空间，在Java8，不是在堆内分配的，它的大小是依赖于本地内存大小！

2019-07-09

作者回复

感谢Xiao同学的提醒。

我想你说的内存模型应该是指Java内存模型（JMM）吧。这里的JVM内存模型跟Java内存模型是不一样的，这里的JVM内存模型和和内存结构是一个意思。

元空间是分配的本地内存，文中开始描述不清楚（已纠正），但后面有明确说明。

2019-07-09

我又不乱来 5

String a="b"应该会放在字符串常量池中。
String c= new String("b") 首先应该放在 堆中一份，再在常量池中放一份。但是常量池中有b了。
第一次留言。不知道理解的对不对。超哥

2019-07-09

作者回复

正确

2019-07-09

Liam 4

请教一个问题，所以1.8开始，方法区是堆的一部分吗？也即是说，方法区的大小受限于堆

2019-07-09

作者回复

方法区不是堆的一部分，方法区和堆存在交集。方法区的静态变量和运行时常量池存放在堆中，但类的元信息等还是存放在了本地内存中。

2019-07-09

Gred 3

老师，运行时变量应该都在方法区中，从java7开始只有字符串常量池移到堆中而已

2019-08-02

作者回复

严格来说，是静态常量池和运行时常量池，静态常量池是存放字符串字面量、符号引用以及类和方法的信息，而运行时常量池存放的是运行时一些直接引用。

运行时常量池是在类加载完成之后，将静态常量池中的符号引用值转存到运行时常量池中，类在解析之后，将符号引用替换成直接引用。

这两个常量池在JDK1.7版本之后，就移到堆内存中了，这里指的是物理空间，而逻辑上还是属于方法区（方法区是逻辑分区）。

2019-08-02

东方奇骥 2

老师，问一下，1.8静态变量和常量存储在的堆里面，那元空间里是什么？文中说之前永久带类的数据存储在了元空间，不是很理解，

2019-07-12

作者回复

元空间主要存储类的一些信息，包括方法、字段、类等描述类信息。

2019-07-17

帽子丨影

1

元空间的存储位置时本地内存，请问下本地内存是个什么东西，在第一张图里没找到啊。

2019-08-20

作者回复

本地内存是一种非JVM堆内存

2019-09-08

发条橙子 。

1

老师，这句话怎么理解

之前永久代的类的元数据存储在了元空间，永久代的静态变量（class static variables）以及运行时常量池（runtime constant pool）则跟 Java7 一样，转移到了堆中。

方法区的一部分是由永久代实现的，永久代主要存储类的静态数据以及运行时常量池并储存在堆内存中。 但是由于容易发生permen内存溢出，后来就发明了元数据空间。那我理解元空间除了存储之前方法区的类信息还包括之前放在永久代中的 静态变量 和 运行时常量池 。
文中为什么说和jdk7一样还是转移到堆中，那不是没有变化么？

2019-07-20

作者回复

是的，没有变化。

2019-07-30

晓杰

1

创建一个线程，就会在虚拟机中申请一个栈帧，这句话有问题吧
应该是创建一个线程，会创建一个栈，然后方法调用一次，就会申请一个栈帧吧

2019-07-11

作者回复

对的，这里是申请一个线程栈。

2019-07-12

Cain

1

常量池在哪个区？堆区？栈区？方法区？静态区？方法区，静态区他俩是什么关系？

2019-07-10

作者回复

在逻辑空间是属于方法区。堆、栈、方法区等，这些是一种规范，是逻辑上的分区。

在物理空间中，常量池是存储在堆内存空间的。

2019-07-12

黑夜里的猫 1

字符串常量不是在java8中已经被放入到堆中了吗，应该不在方法区中了，但是看到老师的图中还在方法区中

2019-07-09

作者回复

方法区是一个规范，并不是一个物理空间，我们这里说的字符串常量放在堆内存空间中，是指实际的物理空间。

2019-07-12

听雨 1

元空间不是本地内存吗，老师说的元空间移入堆内存是什么意思呀，不理解，是元空间属于堆内存的一部分吗？

2019-07-09

作者回复

而到了 Java8，永久代被元空间取代了，元空间存储静态变量...

以上这句话描述不准确。将元空间去掉。元空间是使用的本地内存，在后面讲述到了：“并且元空间的存储位置是本地内存”

2019-07-09

超威、 1

其实常量池中是不会存储具体对象的吧，也是引用，所以说new String的话会现在常量池中去寻找，存在直接由常量池中的引用指向堆中对象，不存在直接开辟新对象？

2019-07-09

作者回复

字符串常量存储在了常量池，引用在运行时存放在了栈中。new String("")是会创建一个新对象的，可以查看一下构造函数：

```
public String(String original) {
this.value = original.value;
this.hash = original.hash;
}
```

2019-07-10

TerryGoForIt 1

老师您好，我想问一下，深入理解 JIT 放到下一节了嘛？我看课程目录 JIT 是在 JMM 之前哇。

2019-07-09

作者回复

是的，调换下位置方便更好理解JIT，因为JIT用到了JVM内存的知识点。声明下，这里不是JMM，JMM是Java Memory Model，而我们这节讲的是JVM的内存模型（Java Virtual Machine St

ructure）。

2019-07-09

谭震弘

0

由于 Java 是多线程语言，当执行的线程数量超过 CPU数量

这里cpu数量不对把，应该是cpu核数？

2019-10-18

作者回复

对的，此处的数量是指CPU核数

2019-10-19

丁浪

0

你的理解跟我不太一致，说下我的理解。
jdk7以前，常量池都是存放在方法区，也就是永久代中。jdk7的时候，只是把“字符串常量池”从方法区/永久代中移除了，“运行时字符串”其实还没变的。jdk8的时候，只是用元空间取代了以前的永久代，“字符串常量池”在堆中，“运行时常量”、类的元数据信息等等都在元空间。元空间属于非堆，使用的是本地内存。

2019-10-17

作者回复

从官方文档解释，class metadata is stored in a new space called Metaspace，所以并没有指出有其他内容存放在元空间。具体的可以参考以下链接：
https://www.oracle.com/webfolder/technetwork/tutorials/mooc/JVM_Troubleshooting/week1/lesson1.pdf

2019-10-19

godtrue

0

课后思考及问题
1：JVM的内存结构
1-1：堆—共享，最大，存几乎所有对象和数组，内部有分为年轻代和老年代，年轻代又继续分为E区、S0区、S1区。
1-2：方法区—共享，是个逻辑分区，8-物理上在堆上分配，存类信息、运行时常量池、静态常量池等信息。8是开辟了部分本地内存，用于存储类信息，常量池信息还在堆上分配。
1-3：虚拟机方法栈—私有，存储线程的方法调用信息，主要是栈帧。
1-4：本地方法栈—私有，存储线程的本地方法调用信息，也是主要是栈帧。
1-5：程序计数器—私有，记录线程的当前执行的位置信息。
老师，请问线程能创建多少，3/4/5中最小的是一个关键限制因素嘛？这三个区域也有对用的配置参数可调控吧？

2019-09-11

作者回复

1、根据机器配置环境创建线程数量不一样；2、3/4/5有可能会成为瓶颈，有参数可以调节，例如，使用-Xss 参数设置栈的大小。

2019-09-11

疯狂咸鱼

0

老师，我看有人说字符串常量池只放引用；那new出来除了堆中会有一个对象，如果字符串常量池没有，也会创建一个，这个对象是在堆中非字符串常量池的地方么

2019-09-02

作者回复

我们说的常量池一般分为静态常量池和运行时常量池，通常字符串常量是存放的引用是在运行时常量池，而字面量是存在了静态常量池。动态生成的字符串，对象是存放在堆中，如果调用intern方法，会将引用存放在常量池中。

2019-09-02

飞翔

0

老师 啥是 本地方法呀？ 可否举个例子

2019-08-12

作者回复

本地方法一般是指非Java实现的代码提供的方法，通常都是C/C++实现的，例如Unsafe下面的allocateMemory

2019-08-13

冉

0

老师你好，我有个疑问，编译器会在.java 文件被编译成.class 文件时，收集 所有类的初始化代码，包括静态变量赋值语句、静态代码块、静态方法，收集在一起成为<clinit>()方法，文中所提 jvm在类初始化阶段会执行<clinit>()方法,那么静态方法是不是也会被一起执行呢？

2019-08-09

作者回复

是的

2019-08-12