

JVM笔记无特殊说明，基于jdk8+HotSpot

[零、参考资料 2](#_Toc37586462)

[一、内存结构 2](#_Toc37586463)

[二、程序计数器(Program Counter Register) 2](#_Toc37586464)

[1概述 2](#_Toc37586465)

[三、Java虚拟机栈(JVM Stack) 3](#_Toc37586466)

[1概述 3](#_Toc37586467)

[2栈帧(Frame) 3](#_Toc37586468)

[2.2局部变量表(Local Variable) 3](#_Toc37586469)

[2.3操作数栈(Operand Stack) 3](#_Toc37586470)

[2.4动态链接(Dynamic Linking) 4](#_Toc37586471)

[2.5方法调用 4](#_Toc37586472)

[四、本地方法栈(Native Method Stack) 4](#_Toc37586473)

[五、 堆(Heap) 4](#_Toc37586474)

[六、方法区(Method Area) 5](#_Toc37586475)

[七、直接内存(Direct Memory) 5](#_Toc37586476)

# 零、参考资料

Java虚拟机规范(Java SE 8版)

Java语言规范基于Java SE 8

深入理解Java虚拟机——JVM高级特性与最佳实践

实战Java虚拟机——JVM故障诊断与性能优化

垃圾回收的算法与实现

深入理解计算机系统

Stackoverflow:

https://stackoverflow.com/questions/16549066/java-major-and-minor-garbage-collections

OpenJdk：

https://openjdk.java.net/jeps/122

http://cr.openjdk.java.net/~sundar/8022483/webrev.01/raw\_files/new/src/share/classes/com/sun/tools/hat/resources/oqlhelp.html

Oracle官方DOC：

https://docs.oracle.com/javase/8/javase-books.htm

https://docs.oracle.com/en/java/javase/13/

https://docs.oracle.com/en/java/javase/14/

https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf

https://docs.oracle.com/javase/specs/jvms/se14/jvms14.pdf

https://docs.oracle.com/javase/specs/index.html

https://www.oracle.com/webfolder/technetwork/tutorials/mooc/JVM\_Troubleshooting/week1/lesson1.pdf

https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

# 一、内存结构

A screenshot of a cell phone

Description automatically generated

PS：蓝色块为线程独享空间

# 二、程序计数器(Program Counter Register)

## 1概述

1.1当前线程正在执行Java方法，记录所执行的虚拟机指令的地址。否则为空(Undefined)。

1.2每个线程都有独立的PC，

1.3字节码解释器工作时通过改变PC的值来选取下一条需要执行的字节码指令

1.4任意时刻JVM线程只会执行一个方法的代码(At any point, each Java Virtual Machine thread is executing the code of a single method)。

1.5这块的容量至少保存一个returnAddress类型的数据或一个与平台相关的本地指针的值(wide enough to hold a returnAddress or a native pointer on the specific platform)

1.6唯一一个在规范中没有规定任何OOM的区域

# 三、Java虚拟机栈(JVM Stack)

## 1概述

1.1每线程都有独立的虚拟机栈，生命周期和线程相同，可以在堆中分配

1.2分配的内存地址可以不需要保证连续

1.3每个方法执行时，都会同步创建个栈帧(Frame)

1.4栈的大小，可以固定，也可以配置动态收缩。

1.5栈的大小为固定时，线程可在创建时自定义栈大小

new Thread(ThreadGroup group, Runnable target, String name, long stackSize)

1.6会抛出StackOverflowError和OutOfMemoryError

## 2栈帧(Frame)

2.1概述

a用于存储数据和部分过程结果的数据结构

b用于处理动态链接(Dynamic Linking),方法返回值(return values for methods)和异常分派(dispatch exceptions)

c生命周期同方法调用过程，且无论方法是否有异常，均视为结束，并销毁

(*A frame is destroyed when its method invocation completes, whether that completion is normal or abrupt (it throws an uncaught exception)*)

d每个栈帧都有自己的一组局部变量表(Local Variable)、操作数栈(Operand Stack)、和指向当前方法所属类的运行时常量池的引用(*a reference to the runtime constant pool of the class of the current method*)

e栈帧数据结构大小仅取决于虚拟机的实现

d栈帧是线程私有数据

## 2.2局部变量表(Local Variable)

2.2.1概述

a长度由编译期决定，并通过方法的code属性(这里)保存以及提供给栈帧使用(*supplied in the binary representation of a class or interface along with the code for the method associated with the frame*)

b一个局部变量槽(Slot)可以保存一个类型为 boolean、byte、char、short、int、float、对象引用(reference)、 returnAddress的数据，long或者double需要两个连续的局部变量槽

c局部变量通过索引(indexing)来定位访问，索引从0开始

d一个局部变量槽的大小，由具体的虚拟机设计决定

e无论读写变量槽是否为原子操作，都不会引起数据竞争和线程安全问题。

f变量槽可以复用

g returnAddress类型目前很少见，现采用异常表来代替

## 2.3操作数栈(Operand Stack)

2.3.1概述

a后进先出(last-in-first-out, LIFO)

b栈深度在编译期决定，并通过方法的code属性保存以及提供给栈帧使用

c全称是：当前栈帧的操作数栈(Where it is clear by context, we will sometimes refer to the operand stack of the current frame as simply the operand stack.)

d在栈帧创建时，操作数栈是空的

e栈中元素的数据类型必须与字节码指令的序列严格匹配

## 2.4动态链接(Dynamic Linking)

2.4.1概述

a通过一个指向当前方法所在类型的运行时常量池的引用，对当前方法的代码实现动态链接(Each frame contains a reference to the run-time constant pool for the type of the current method to support dynamic linking of the method code)

b类文件的常量池中存有大量的符号引用，字节码中的方法调用指令就是以常量池指向方法的符号引用作为参数

b.1静态解析：指一部分符号在类加载阶段或者第一次使用的时候就被转化为直接引用的转化

b.2动态链接：指一部分符号在每一次运行期间转化为直接引用的转化

## 2.5方法调用

2.5.1概述

a正常调用完成是指在方法执行过程中没有抛出任何异常，包括但不限于通过throw语句显式抛出的异常(A method invocation completes normally if that invocation does not cause an exception to be thrown, either directly from the Java Virtual Machine or as a result of executing an explicit throw statement)

b如果有返回值的话，使用哪种返回指令取决于方法返回值的数据类型

c异常调用完成是指方法在执行过程中抛出了异常，并且没有捕获异常

d方法异常调用完成，一定不会有方法返回值返回给调用者(A method invocation that completes abruptly never returns a value to its invoker)

e方法退出后都必须返回到最初方法被调用的位置，程序才能继续执行

f方法退出过程实际上等同于吧当前栈帧出栈

# 四、本地方法栈(Native Method Stack)

1概述

1.1为虚拟机用到的本地(Native)方法服务

1.2如果虚拟机不支持Native方法或不依赖传统栈(通常称C stack)，可以没有本地方法栈

1.3会抛出StackOverflowError和OutOfMemoryError

1.4栈的大小，可以固定，也可以配置动态收缩。

1.5栈的大小为固定时，线程可在创建时自定义栈大小

1.6虚拟机栈和本地方法栈可以被设计为合二为一，比如HotSpot虚拟机。

# 堆(Heap)

1概述

1.1在虚拟机启动时创建，被所有线程共享

1.2可以划分出线程私有的线程缓冲区(Thread Local Allocation Buffer,TLAB)

1.3内存地址不需要保证连续，但在逻辑上应被视为连续的

1.4部分虚拟机设计，会针对大对象(比如数组)存储，要求内存地址是连续的

1.4容量可以固定，也可以动态扩展

1.5存储的对象无法显式销毁(objects are never explicitly deallocated)

1.6会抛出OutOfMemoryError

# 六、方法区(Method Area)

## 1概述

1.1别名非堆(Non-heap),不等价于永久代(Permanent Generation),被所有线程共享

1.2从JDK8开始，在本地内存中实现的元空间(Metaspace)代替永久代实现方法区

1.3存储了每个类的结构信息(It stores per-class structures),比如，运行时常量池(the run-time constant pool)，字段和方法数据(field and method data)，构造函数和普通方法的字节码内容(the code for methods and constructors)，还包括一些在类、实例、接口初始化时用到的特殊方法(*including the special methods used in class and instance initialization and interface initialization.*)

1.3容量可以固定，也可以动态扩展，可以不实现GC

1.4会抛出OutOfMemoryError

## 2运行时常量池(Run-Time Constant Pool)

2.1概述

a是class文件中每个类或接口的常量池表(constant\_pool)的运行时表现形式(a per-class or per-interface run-time representation of the constant\_pool table in a class file)

b包含编译期已知的数值字面量和运行期解析后获取的方法或字段引用(*ranging from numeric literals known at compile-time to method and field references that must be resolved at run-time*)

ps:String.intern()

c会抛出OutOfMemoryError

d虚拟机规范没有对此作任何细节的要求

# 七、直接内存(Direct Memory)

## 1概述

1.1不是虚拟机运行时数据区的一部分，也不是《Java虚拟机规范》中定义的内存区域

1.2可能会抛出OutOfMemoryError

1.3可以通过DirectByteBuffer对象作为这块内存的引用进行操作

# 八、类(Class)

## 1概述

1.1类文件格式不是必须存在于文件里。

1.2多字节数据项必须严格按照大端在前(big-endian)顺序进行存储(*Multibyte data items are always stored in big-endian order,where the high bytes come first*)

1.3 Class文件是一组以8字节为基础单位的二进制流，各个数据项严格按照顺序紧凑排列，没有任何分隔符

1.4Class文件格式采用一种类似C语言结构体的伪结构(This chapter presents the class file format using pseudostructures written in a C-like structure notation)，只有两种数据类型:无符号数(unsigned)和表(table)

1.5无符号数是基本的数据类型，可以用来描述数字、索引引用、数量值或者按照UTF-8编码构成字符串值

1.6表由多个无符号数或其他表作为数据项构成的复合数据类型

1.7当描述同一类型但数量不定的多个数据时，经常会使用一个前置的容量计数器加若干个连续的数据项的形式，这时候称这一系列连续的某一类型的数据为某一类型的集合

## 2类结构

|  |  |  |
| --- | --- | --- |
| 类型 | 数量 | 说明 |
| u4 | 1 | 魔数(magic)，确定是否为虚拟机所接受的类文件，固定值为0xCAFEBABE |
| u2 | 1 | 副版本号(minor\_version)  1、主版本号为45-55时，副版本号可以为任何值  2、从主版本号56(jdk12)开始，副版本号必须为0或65535  3、当副版本号为65535时，为预览版 |
| u2 | 1 | 主版本号(major\_version)，  1、值从45开始  2、即使文件格式无任何变化，虚拟机也必须拒绝执行类主版本号超过虚拟机版本号的文件。 |
| u2 | 1 | constant\_pool\_count; |
| cp\_info |  | constant\_pool[constant\_pool\_count-1]; |
| u2 | 1 | access\_flags; |
| u2 | 1 | this\_class; |
| u2 | 1 | super\_class; |
| u2 | 1 | interfaces\_count; |
| u2 |  | interfaces[interfaces\_count]; |
| u2 | 1 | fields\_count; |
| field\_info | fields[fields\_count] | ; |
| u2 | 1 | methods\_count; |
| method\_info | methods[methods\_count] | ; |
| u2 | 1 | attributes\_count; |
| attribute\_info | attributes[attributes\_count] |  |