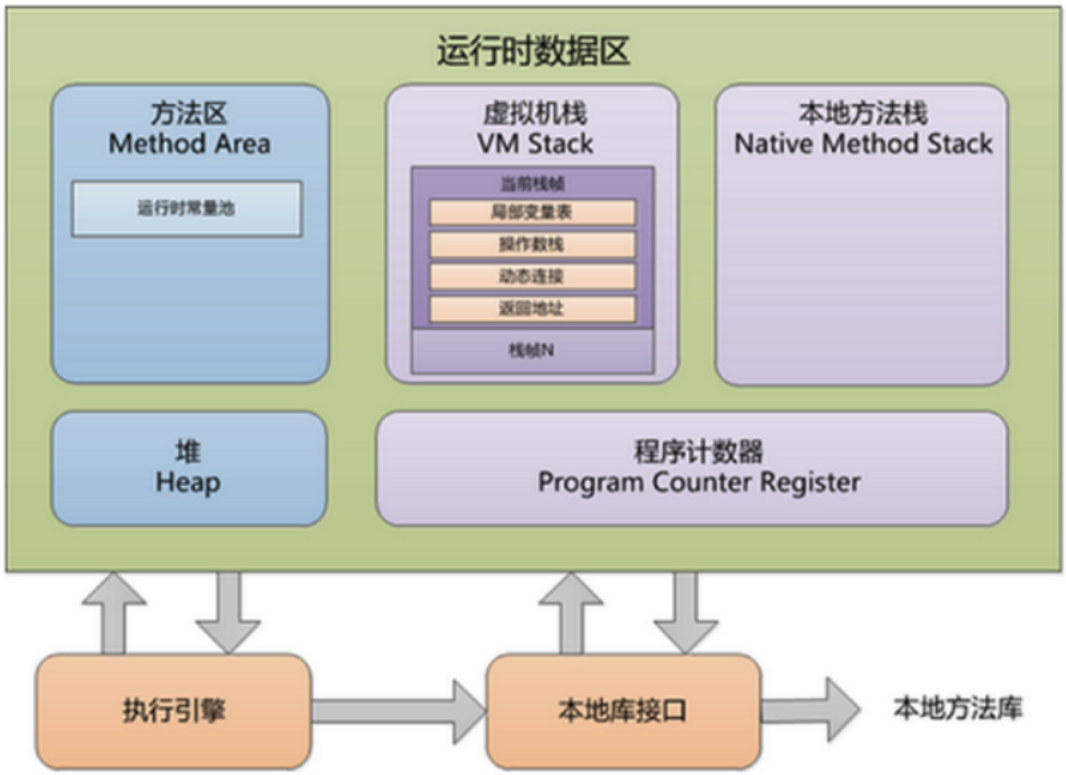
jvm

1.学习推荐

- 视频推荐：图灵学院的直播课 jvm专栏
- 书推荐：周志明所著《深入理解java虚拟机》

2.jvm的内存模型的理解

2.1 图解



2.2 图解含义

1. jvm中有三个组件：类装载子系统，字节码执行引擎，运行数据区
2. 运行数据区：

- 共享区 : 堆，方法区（运行时常量池）
- 非共享区: 栈，程序计数器，本地方法栈

3. 程序计数器：

- 定义：是代码执行位置的一个指示器
- 特点：每一个线程都会拥有一个程序计数器，与线程声明周期相同
- 作用：用于循环，分支，异常处理，线程回复等；

4. 栈：

- 定义：一个线程会创建一个线程栈，栈以栈帧为单位，没创建一个方法都会在线程栈中开辟一个栈帧,栈帧中由局部变量表，操作数栈，动态连接，方法出口，局部变量表中存的基本类型和引用类型，是以变量槽为单位的，long和double占两个变量槽

5. 堆：

- 定义：用来存储应用类型的数据，堆的结构也很复杂，不同的jvm版本将堆模型版本划分，一般分为

两种，年代划分，分区划分

- 年代划分：老年代和新生代，新生代：eden,surrvier区 ： from to区 8.1.1
- 分区：按照页面划分，G1开始--->后面会详细介绍堆

6. 方法区：

- 定义：用于存储类变量，常量数据，常量区里面有运行时常量区
- 运行时常量区：举例：String::inter()方法，如果，方法区已经有这个常量，直接获取过来，没有，就会创建放回常量区，不过后续jvm版本方法区合并到堆中，又有了不同的见解。

3.jvm如何创建对象和访问

3.1 创建对象的流程

jvm在创建对象的时候经过这么几个步骤

1. 类加载
2. 验证
3. 准备

- 分配内存: 这个时候分配内存，jvm呢的堆内存一般为两种，一种碎片内存，一种连续内存，当线程来申请内存时，会出现并发安全问题
- 怎么解决：jvm是通过CAS或者ThreadLocal来解决的
- CAS：比较在交换，请看我后续的并发章节
- ThreadLocal: 称之为本地线程机制：作用：我们可以将数据存入到ThreadLocal中，同一个线程是数据共享的，不同线程是数据隔离的，每一个线程都会创建一个ThreadLocal，实际上它是一个map，key代表的是当前线程，value就是我们存入的数据

4. 初始化对象

- 对象头，实例数据等

3.2 对象的理解

1. 对象一般分为三部分：对象头，实例数据，对齐填充
2. 对象头:标记位markword,类型指针

- 在32位的对象头中：25hash,4位代表分代年龄，2位代表锁标记，1位代表是否偏向锁
- 64中：31hash， 25空闲，,4位代表分代年龄，2位代表锁标记，1位代表是否偏向锁，1位空闲
- 类型指针：用于指向数据实例数据，或者类型数据的指针，如果是数组，还有一个单独的长度

3. 实例数据：代表我们实际要初始化数据的部分

4. 对齐填充：没什么含义

3.3 如何访问对象

1. java规范给jvm定义了一套规范，就和数据库一样，我定义一套标准，你们要开发你们去实现，所以如何访问对象也是jvm的范畴
2. 访问方式：句柄池和直接访问

- 句柄池: jvm会在堆中维护一个句柄池，用于存放类型数据指针和实例数据指针，类型数据指针用于访问方法区中的，而实例用于访问堆中的实例数据
- 直接访问: 当然类型数据指针不变，没有了实例数据指针直接访问(采用)

4.如何排查线上oom问题

排查线上oom，需要先知道jvm中那些地方会触发oom

4.1 栈

- 栈一般分为两种情况：stackoverflow 和 oom。
- stack-over-flow: 当我们开启线程的时候虚拟机会为线程分配，栈空间，和程序技术器，栈的深度虚拟机已经为起指定了最大深度，举个列子：我们在栈中不停的递归调用方法，这就会导致不停的开辟栈帧，导致栈空间不足，发生stackoverflow这样的异常。
- oom: 它的原因是jvm内存不足，举个列子：我们定义一个循环创建线程的程序，这个时候jvm就会不停的给线程分配线程栈，当达到一定的数量之后，jvm不够给足够的空间，就会触发oom异常。

4.2 堆

- 堆也分为两种情况，但都是oom异常，但原因不一样：内存溢出，内存泄漏
- 内存泄漏：我们一般需要通过工具查看GCroots引用链，很有可能是引用错误导致，堆中的对象不能被垃圾回收器自动回收掉，触发oom
- 内存溢出：本该对象就需要存活，可能是对象太大，或者生命周期太长，这个时候我们要通过参数调整堆的内存大小来解决问题,当然具体是提高堆中的那块内存大小，这就需要分析了。

4.3 方法区

- 方法区发生oom主要是因为，方法区中的字符串存入的比较多，导致方法区的内存不够，触发oom比如你不停的给set集合中添加字符串，就会报oom。

4.4 直接内存

- 直接内存不受限于jvm内存，它不属于jvm部分，但是一块经常被使用的数据区，它受限于你的主机内存，如果主机内存不能够支撑，就会触发oom

5.gc垃圾回收机制

5.1 gc垃圾回收的基础概念

1. 什么是垃圾

我们将被GCroots集合引用的对象称之为非垃圾对象，反之就是垃圾

2. 什么是GCroots

GCroots集合是由：线程栈中的局部变量，方法区中的类变量，等总之要引用堆中的对象，这些组成的集合称之为GCroots

3. 怎么寻找垃圾

- 引用计数法：这个方法就是数对象的引用指针个数，为0垃圾，不为0不是垃圾，这会造成误判，存在循环引用问题
- 可达性分析算法：根据GCroots找响应的可达对象，对象可达，代表非垃圾，不可达就是垃圾

4. 三种基本算法

- 标记–清除算法：就是根据可达性算法标记非垃圾对象，将其他的垃圾对象进行清除：缺点：会造成内部碎片，导致我们下次要装入一个大的对象，明明内存空间够，结果放不进去。
- 标记–整理算法：就是根据可达性算法标记非垃圾对象，将存活的对象移到一边，其他垃圾对象一次性清除掉，缺点：整理的时候会触发gc线程STW，比较慢。
- 标记–复制算法：就是根据可达性分析标记存活的对象，但是我们这次内存一分为2一般应用，一般用于复制存活的对象到另一边，然后一次性清除垃圾对象。

5. STW stop the world

触发STW的时机一般有，GCroots标记的时候，注意这里可以指的是扫描的一个对象，在CMS,G1,ZGC中有所体现。还有就是我们在标记整理的时候会触发STW。

6. 并发和并行

在jvm中：并发值得是不同线程一起执行，例如 gc线程和用户线程一块执行，并行指的是相同线程：比如gc多线程回收。

5.2 gc垃圾回收器的分代6种

1. 堆的分代模型

- 分代：堆被分为老年代和新生代，新生代被分为eden区和surrver区，survicer区又被分为 from 和to区，我们新生的对象一般会在eden区，触发minorgc进行回收,回收不掉的对象会转到from区，下一次回收会在from 和to 采用标记–复制算法（存活少的对象）去迭代年龄，当年龄大于15或者某个对象的内存大于一半,会直接被丢进老年代，老年代采用标记–整理算法（存活多的对象）回收。
- 6种分代垃圾算法：serial 和 serial old 分别都是单线程gc通过不同的算法回收新生代和老年代；parnew:值得是新生代的gc线程并行执行;Parallel 和Paraller Old指的是提搞了gc的吞吐量，gc并行于年轻代和老年代；CMS垃圾回收器，倾向于减少STW的触发时间，会经历这么几个阶段：初始标记–––>并发标记––––>重新标记––––––>并发清除。这几个阶段，只有初始和重新标记的时候才会STW。

5.3 gc垃圾回收器的分区4种

在这里我知道就是两种，如果想要了解更多，可以去查看我推荐的书籍

- G1:g1垃圾回收器可以说是垃圾回收期的一个里程碑，它的模型不在采用分代，而是先相等分区，然后在进行对每个区域进行分代，这样可以更精确的回收那块区域，而不是回收整个堆，回收过程采用了：初始标记–––>并发标记––––>重新标记––––––>并发清除这几个阶段。
- ZGC：说ZGC之前我们得先了解一个叫着色指针技术：在64位虚拟地址中，对象头25位空闲，31位hash,4位分代，2位锁标记，1位偏向锁标记，1位空闲，将25位称之为高位，其他称之为地位，高位空闲我们可以利用起来，三个bit位表示指针颜色 100，001，010代表不同的颜色。zgc的时候比g1阶段更加复杂：初始标记–––>并发标记––––>重新标记––––––>并发转移准备––––––>初始转移––––––––>并发转移。当然zgc这些过程有很多细节点，怎么转移的，内存如何做的。不做介绍了，可以看书学习。

6.jvm的类加载机制

6.1 jvm的类加载过程

- 加载阶段：jvm 将字节码文件的静态存储结构转换为jvm运行数据区的动态存储结构的过程称之为加载。
- 验证阶段：验证意思就是jvm会判断你的字节码文件是否符合规范，防止自身的进程遭到破坏，也是一种自我保护机制。
- 准备阶段：这个阶段会定义变量（类变量），会分配内存，初始化静态变量，这里一般情况下给的是默认值，特殊情况下：比如 static final修饰时，就会给一个不在变的固定值，这也是区分实例变量和类变量的一个关键点。试分析以下，static变量在准备阶段已经初始化，而实例变量在初始化阶段，准备阶段的后面，自然，实例变量可以调用类变量，类变量中不能用实例变量。

- 解析阶段：jvm会对字节码文件进行，方法，接口，类，变量等解析，是否符合语法规范，如果出错就抛出异常。
- 初始化阶段：这个阶段一般是子类调用父类，父类初始化，反射，new,main启动时的主类等会触发初始化，初始化才是真正给变量赋值的一个阶段。

6.2 双亲委派机制

- 类加载器的一个介绍：jvm中提供了两种类型的类加载器，一种是基于c实现的bootstrapClassloader,另一个种是基于 java实现的classLoader;bootstrapClassLoader:用于加载jre/lib文件的类库，classLoader分为两种，一种是 extClassLoader(加载jre/lib/ext)和AppClassLoader(加载classpath)
- 双亲委派机制

源代码角度分析 Java 复制代码

```java
protected Class<?> loadClass(String name, boolean resolve){
    // First, check if the class has already been loaded
    Class<?> c = findLoadedClass(name);  //看本地加载过没

    if (parent != null) {//判断父类加载器是否为null
        c = parent.loadClass(name, false);  不为调用即可
    } else {
        c = findBootstrapClassOrNull(name);  为null调用bootstrap
    }
    //父类加载到了？没加载到我自己加载
    if (c == null) {
        // If still not found, then invoke findClass in order
        // to find the class.
        long t1 = System.nanoTime();
        c = findClass(name);
    }
    return c;
}
:上述只是方法中的关键代码
```

根据上述代码我们不难画出一个双亲委派机制的图，图略：先看自己加载过没，没有抛给父类，父类先加载，加载不到，在向上抛，直到顶级父类加载不到，在往下抛依次加载，直到加载到为之，注意这里的父子类没有父子关系，只是这么说而已，举个列子：我要自定义一个Object类，按照上述加载顺序我自己的肯定是加载不到的，这也是java类库的一种自我保护。如何打破：就自定义一个classloader重写findclass()方法即可。

7.字节码执行引擎的一个作用

字节码执行引擎：对于一个线程而言，线程栈中的所有方法都是运行态，而对于字节码执行引擎而言，它只关注栈顶部的栈帧，用于操作程序计数器。