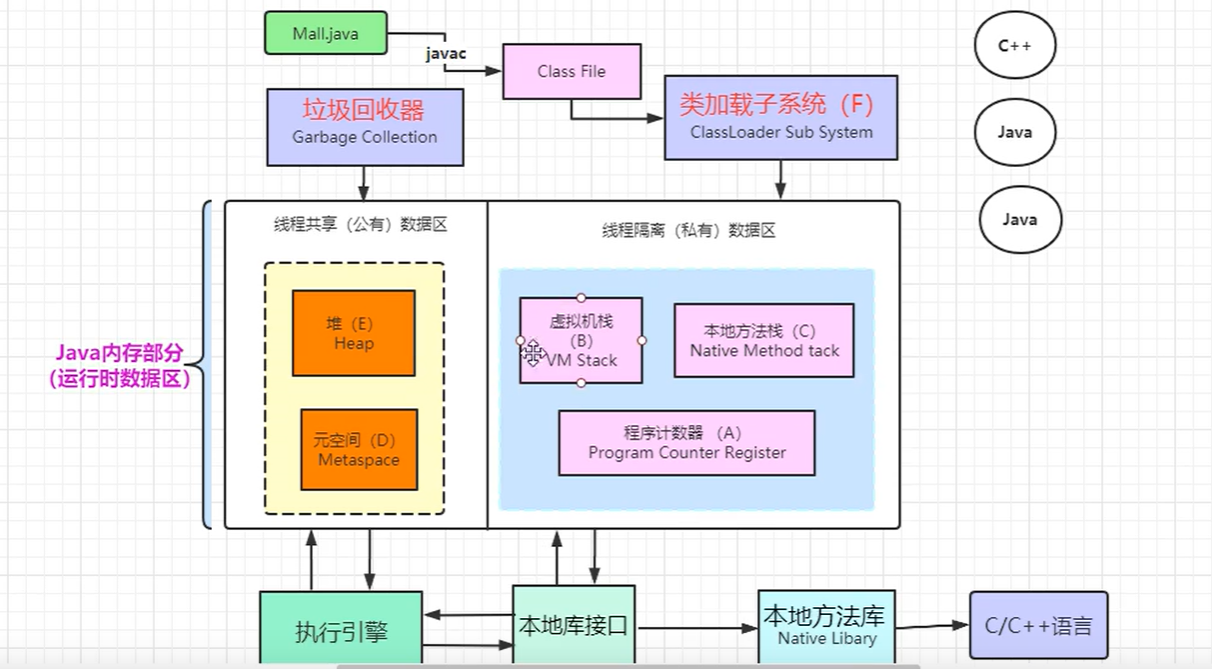
****

**JVM 运行时数据区**

程序计数器：记住下一条jvm指令的执行地址，线程私有的

虚拟机栈（线程私有）

栈：每个线程运行时需要的内存空间

栈帧：每个方法运行时需要的内存空间，栈由多个栈帧组成

活动栈帧，对应着当前正在执行的那个方法

存储方法的局部变量表、操作数栈、动态连接和方法返回地址等信息。

本地方法栈：Java虚拟机调用本地方法时提供的内存空间

堆（线程共享）：存储java对象实例，有垃圾回收机制

方法区：方法区存放的数据是被类加载器加载后的类信息，运行时常量池等

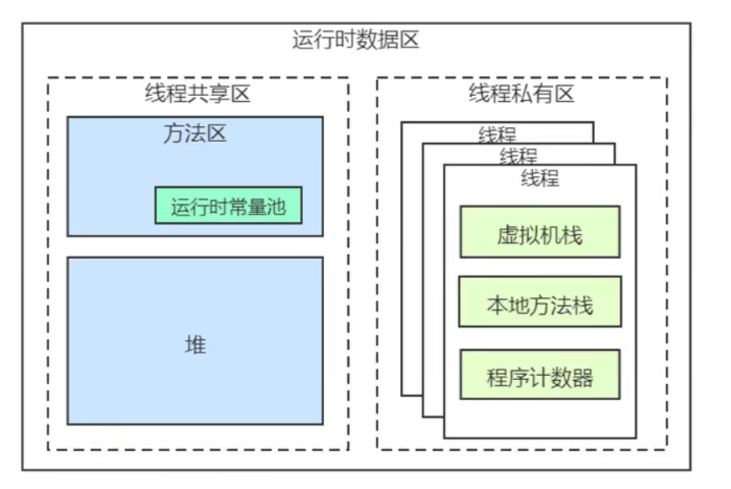
注：

操作数栈：用于保存计算过程的中间结果，作为变量的临时存储空间

方法返回地址：调用该方法的指令的下一条指令的地址.

动态连接：指向运行时常量池的方法引用

运行时常量池：常量池是\*.class文件中的，当该类被加载，他的常量池信息就会放入运行时常量池，并把里面的符号地址变为真实的地址。



**堆栈的区别**

堆是运行时确定内存大小，栈是编译时确定内存大小

堆用来存储java对象实例，栈用来存储局部变量和方法调用

堆是线程公有的。栈是线程私有的。

栈的空间大小远远小于堆的

堆空间不足：OutOfMemoryError，栈空间不足：StackOverFlowError

**如何判断对象可以回收**

**引用计数法，循环引用**

**可达性分析算法，扫描堆中的对象，看是否能够沿着GC Root对象为起点的引用链找到该对象，找不到，表示可以回收**

**哪些对象可以作为GC Root？（根对象，不可能被回收的对象）**

**1.虚拟机栈（栈帧的局部变量表）所引用的对象；**

**2.本地方法栈中 JNI（Native 方法）引用的对象**

**3.方法区的静态变量、常量所引用的对象**

**四种引用**

**强引用，创建一个对象并把这个对象赋给一个引用变量。（如果有GCRoot的强引用）垃圾回收器不会回收它**

**软引用，内存不足时（自动触发GC），会被回收**

**弱引用，无论内存是否充足，只要进行GC，都会被回收。发现即回收**

**虚引用，和没有任何引用一样，随时可能会被回收，虚引用的主要作用是跟踪对象被垃圾回收的状态**

**垃圾回收概念**

**垃圾回收就是**自动回收不再使用的对象和资源，释放内存

**垃圾回收算法**

**标记清除算法，标记无用对象，然后进行清除回收。速度快，容易产生内存碎片**

**标记整理算法，将存活对象都向一端移动，然后清理掉边界以外的内存。不会产生内存碎片，由于整理要移动对象，导致效率较低**

**复制算法，将内存划分为大小相等的两块，每次只使用其中的一块，当这一块用完了，就将存活对象复制到另一块上面，然后清理掉已使用过的那块。（然后再把已使用过的内存空间清理掉。）不会产生内存碎片，要占用双倍的内存空间**

**分代收集算法，根据对象存活时间将内存划分为新生代和老年代，新生代使用复制算法，老年代使用标记清除算法或者标记整理算法**

**分代垃圾回收**

**有两个分区：新生代和老年代，新生代包括**eden、from、to

**1.对象首先分配在**eden**区**

**2.新生代空间不足时，触发 minor gc，**eden**区和 from区存活的对象复制到to区中，存活的对象年龄+1并且交换 from、to**

**3.minor gc 会引发STW（stop the world）暂停其它用户的线程，等垃圾回收结束，用户线程才恢复运行**

**4.当对象寿命超过阈值15时，会晋升至老年代**

**5.当老年代空间不足，会先尝试触发 minor gc，如果之后空间仍不足，那么触发 full gc，STW的时间更长**

**注：如果对象大小大于新生代的总大小，则直接存入老年代中。**大对象也会直接进入老年代。

**垃圾回收器**

**Serial GC：**

**特点：单线程，适用于单核CPU。**

**使用场景：客户端应用或小型服务器。**

**Parallel GC：**

**特点：多线程，适用于多核CPU。**

**使用场景：需要高吞吐量的应用。**

**CMS（Concurrent Mark-Sweep） GC：**

**特点：并发执行，减少停顿时间。**

**使用场景：对响应时间敏感的应用。**

**G1（Garbage-First） GC：**

**特点：将堆分为多个区域，优先回收垃圾最多的区域。**

**串行：只会使用一个CPU或者一条GC线程进行垃圾回收，并且在垃圾回收过程中暂停其他工作线程。**

**吞吐量优先：多线程，堆内存较大，多核CPU，让单位时间内，STW的时间最短 0.2 0.2 0.4 采用复制算法实现。**

**响应时间优先：多线程，堆内存较大，多核CPU，尽可能让单次STW的时间最短 0.1 0.1 0.1 0.1 0.1 0.5 使用标记-清除算法实现**

**垃圾回收的触发条件**

**新生代回收（Minor GC）：新生代空间不足时触发**

**老年代回收（Major GC）：老年代空间不足**

**整个堆回收(年轻代+老年代+元空间) Full GC：老年代空间不足、元空间不足(如加载过多类)、长时间没有进行过 Full GC、调用 System.gc()、内存溢出（OutOfMemoryError）**

**调优建议**

**堆大小：根据应用需求调整堆大小，避免频繁GC。**

**选择合适的GC算法：根据应用特点选择适合的垃圾回收器。**

**监控GC日志：通过GC日志分析性能瓶颈，优化内存使用。**

**G1（Garbage First）**

**直接将堆分成多个rigion区域。每个区域都可以充当 Eden 区、Survivor 区或者old区中的一个。G1 会将超过 region 50% 大小的对象归类为 Humongous 对象，并放置在相应的 region 中。**

**G1采用的是标记 - 整理算法，优先回收死亡对象较多的区域**

G1的垃圾回收阶段**：**

1.新生代GC

当Eden区内存空间耗尽的时候，G1会启动一次年轻代垃圾回收.

根扫描，GC Roots连同RSet记录的外部引用作为扫描存活对象的入口。 (对于年轻代的Region，它的RSet 只保存了来自老年代的引用) 初始标记

更新&&处理 RSet，处理RSet的信息并且扫描，将老年代对象持有年轻代对象的相关引用都加入到GC Roots下，避免被回收掉

复制对象，把扫描之后存活的对象往空的Survivor区或者老年代存放，其他的Eden区进行清除

注：因为Minor GC 是回收年轻代的对象，但如果老年代有对象引用着年轻代，那这些被老年代引用的对象也不能回收掉

2.新生代GC+并发标记

第二个阶段 Young Collection + Concurrent Mark，即新生代的垃圾收集同时执行一些并发的标记。

全局并发标记 初始标记（STW）、并发标记、最终标记（STW）以及清理（STW）

a.初始标记----标记GC Roots能直接关联到的对象，并且会触发一次Young GC，stop the world

b.并发标记----标记GC Roots可达性对象，并收集region存活对象信息，计算每个region的存活对象的比例

计算每个region的对象活性（该region存活对象的比例，G1垃圾回收的时候并不是所有region都会参与回收的，根据回收的价值高低来优先回收价值较高的region）

c.最终标记----标记在并发阶段因用户程序发生变化的对象，stop the world

d.清理----对各region的回收价值和成本排序，根据用户期望GC停顿时间进行回收，将回收效益高的Region放入Collect Set中去，stop the world

(这个阶段并不会实际去做垃圾的回收，也不会执行存活对象的拷贝)

3.混合回收

Mixed GC会选定所有的年轻代Region，部分回收价值高的老年代Region（回收价值高其实就是垃圾多）进行采集

将Collect Set中的Region中的存活对象拷贝到其他空的Region中去，原本的Region进行清除

**Java会存在内存泄漏吗？请简单描述**

内存泄漏是指不再被使用的对象或者变量一直被占据在内存中。

可能存在。创建了一个对象，以后一直不再使用这个对象，然后这个对象却一直被引用，即这个对象无用但是却无法被垃圾回收器回收的。

**G1和CMS的区别**

G1采用标记-整理算法,CMS采用标记-清除算法,所以G1不会产生很多垃圾碎片.

**垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？**

GC采用有向图的方式记录和管理堆中的所有对象。当GC确定一些对象为"不可达"时，GC会回收这些内存空间。

可以。程序员可以手动执行System.gc()

**说一下 JVM 有哪些垃圾回收器？**

Serial（串行GC）-复制

ParNew（并行GC）-复制

Parallel Scavenge（并行回收GC）-复制

Serial Old（MSC）（串行GC）-标记-整理

CMS（并发GC）-标记-清除

Parallel Old（并行GC）--标记-整理

G1（JDK1.7update14才可以正式商用）

1~3用于新生代垃圾回收：新生代的垃圾回收称为minor GC

4~6用于老年代垃圾回收（当然也可以用于方法区的回收）：老年代的垃圾回收称为full GC

G1独立完成"分代垃圾回收"

**新生代垃圾回收器和老年代垃圾回收器都有哪些？有什么区别？**

新生代回收器：Serial、ParNew、Parallel Scavenge

老年代回收器：Serial Old、CMS、Parallel Old

整堆回收器：G1

新生代垃圾回收器一般采用的是复制算法，老年代垃圾回收器一般采用的是标记-整理的算法。

新生代：新生代是堆中的一块内存空间，新生代主要存放新创建的对象，垃圾回收会比较频繁

老年代：老年代主要存放JVM认为生命周期比较长的对象（经过几次的Young Gen的垃圾回收后仍然存在），垃圾回收没有那么频繁

**类加载阶段**

1. 加载：将类的字节码载入方法区中，如果这个类还有父类没有加载，就先加载父类

2. 链接

验证：验证类是否符合JVM规范，安全性检查

准备：为static变量分配空间，设置默认值

如果static变量是final的基本类型，赋值在准备阶段完成

如果static变量是final的引用类型，赋值在初始化阶段完成

解析：将常量池中的符号引用解析为直接引用

3. 初始化

初始化，调用类的构造方法cinit

发生的时机：

main方法所在类，总是被首先初始化

首次访问这个类的静态变量或静态方法时

子类的初始化，如果父类还没初始化，会引发父类的初始化

子类访问父类的静态变量，只会触发父类的初始化

Class.forName

new关键字会导致初始化

**什么是类加载器，类加载器有哪些**

类加载器就是把类文件加载到虚拟机中

启动类加载器：JAVA\_HOME/jre/lib （加载哪的类）

扩展类加载器：JAVA\_HOME/jre/lib/ext

应用程序类加载器：classpath

自定义类加载器：通过继承 java.lang.ClassLoader类的方式实现

**双亲委派模型**

当一个[类加载](https://so.csdn.net/so/search?q=%E7%B1%BB%E5%8A%A0%E8%BD%BD&spm=1001.2101.3001.7020)器收到类加载任务时，会先交给自己的父加载器去完成，因此最终加载任务都会传递到最顶层的BootstrapClassLoader，只有当父加载器无法完成加载任务时，才会尝试自己来加载。

作用：1. 避免类的重复加载。2. 保证类的加载安全性。3.提高类加载的效率。

a.一旦类被父类加载器加载，子类加载器不会再重复加载同名类

b.用户伪造同名类，假设用户编写了一个 `java.util.HashMap` 类，并试图替换JDK原生的实现

public class HashMap {

public HashMap() {

System.out.println("恶意HashMap被加载！");

}

}

加载过程分析：

1. 用户程序尝试加载自定义的HashMap。

2. 应用类加载器委派给扩展类加载器，再委派给启动类加载器。

3. 启动类加载器 从 rt.jar中找到了HashMap类并直接加载。

4. 结果：用户自定义的 HashMap永远不会被加载，因为父类加载器已完成了加载。用户的同名类被忽略，核心类未被篡改

**G1的垃圾回收阶段：**

1.新生代GC

当Eden区内存空间耗尽的时候，G1会启动一次年轻代垃圾回收.

根扫描，GC Roots连同RSet记录的外部引用作为扫描存活对象的入口。

复制对象，把扫描之后存活的对象往空的Survivor区或者老年代存放，其他的Eden区进行清除

2.新生代GC+并发标记

全局并发标记 初始标记（STW）、并发标记、最终标记（STW）以及清理（STW）

a.初始标记----标记GC Roots能直接关联到的对象，并且会触发一次Young GC，stop the world

b.并发标记----标记GC Roots可达性对象

c.最终标记----标记在并发阶段因用户程序发生变化的对象，stop the world

d.清理----将回收效益高的Region放入Collect Set中去，stop the world

3.混合回收

清理Collect Set中的Region

**说一下 JVM 调优的工具？**

JDK 自带了很多监控工具，都位于 JDK 的 bin 目录下，其中最常用的是

jconsole 和 jvisualvm 这两款视图监控工具。

jconsole：用于对 JVM 中的内存、线程和类等进行监控；

jvisualvm：JDK 自带的全能分析工具，可以分析：内存快照、线程快照、程序死锁、监控内存的变化、gc 变化等。

**常用的 JVM 调优的参数都有哪些？**

-Xms2g：初始化推大小为 2g；

-Xmx2g：堆最大内存为 2g；

-XX:NewRatio=4：设置年轻的和老年代的内存比例为 1:4；

-XX:SurvivorRatio=8：设置新生代 Eden 和 Survivor 比例为 8:2；

–XX:+UseParNewGC：指定使用 ParNew + Serial Old 垃圾回收器组合；

-XX:+UseParallelOldGC：指定使用 ParNew + ParNew Old 垃圾回收器组

合；

-XX:+UseConcMarkSweepGC：指定使用 CMS + Serial Old 垃圾回收器组

合；

-XX:+PrintGC：开启打印 gc 信息；

-XX:+PrintGCDetails：打印 gc 详细信息。

**对象创建的几种方式**

new出一个对象

利用反射创建对象：获取该Class对象、利用该Class对象的getConstrutor方法来获取指定的构造器、调用Construtor的newInstance方法来创建Java对象。

利用反序列化创建对象

clone创建对象

**对象创建的主要流程**

虚拟机遇到一条new指令时，先检查常量池是否已经加载相应的类，如果没有先进行类加载。接下来分配内存。若Java堆中内存是绝对规整的，使用“指针碰撞“方式分配内存；如果不是规整的，就从空闲列表中分配，叫做”空闲列表“方式。划分内存时还需要考虑一个问题-并发，也有两种方式: CAS同步处理，或者本地线程分配缓冲。然后内存空间初始化操作，接着是做一些必要的对象设置(元信息、哈希码…)，最后执行方法。



**为对象分配内存**

指针碰撞：如果Java堆的内存是规整的，即所有用过的内存放在一边，空闲的放在另一边。分配内存时把位于中间的指针指示器，向空闲的内存移动一段与对象大小相等的距离。

空闲列表：如果Java堆的内存不是规整的，则需要由虚拟机维护一个列表来记录哪些内存是可用的，在分配的时候可以从列表中查询到足够大的内存分配给对象，并在分配后更新列表记录。

**处理并发安全问题**

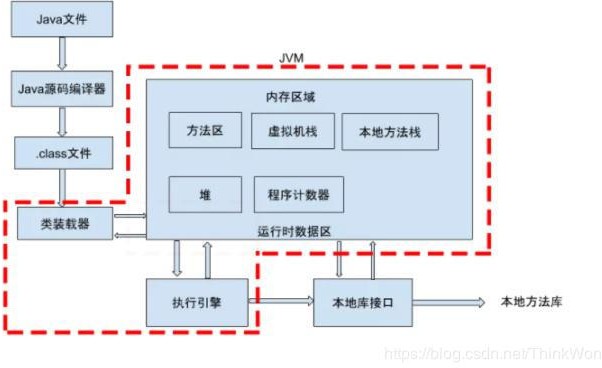
可能出现正在给对象 A 分配内存，指针还没来得及修改，对象 B 又同时使用了原来的指针来分配内存的情况。

1.对分配内存空间的动作进行同步处理

2.每个线程在 Java 堆中预先分配一小块内存，称为本地线程分配缓冲（Thread Local Allocation Buffer,TLAB）。哪个线程要分配内存，就在哪个线程的TLAB 上分配。只有 TLAB 用完并分配新的TLAB 时，才需要同步锁。通过-XX:+/-UserTLAB参数来设定虚拟机是否使用TLAB

**说一下 JVM 的主要组成部分及其作用？**

Class loader(类装载)、Execution engine(执行引擎)；两个组件为Runtime data area(运行时数据区)、Native Interface(本地接口)

作用 ：首先通过编译器把 Java 代码转换成字节码，类加载器再把字节码加载到内存中，将其放在运行时数据区的方法区内，命令解析器执行引擎（Execution Engine），将字节码翻译成底层系统指令，再交由 CPU 去执行，而这个过程中需要调用其他语言的本地库接口（Native Interface）来实现整个程序的功能。