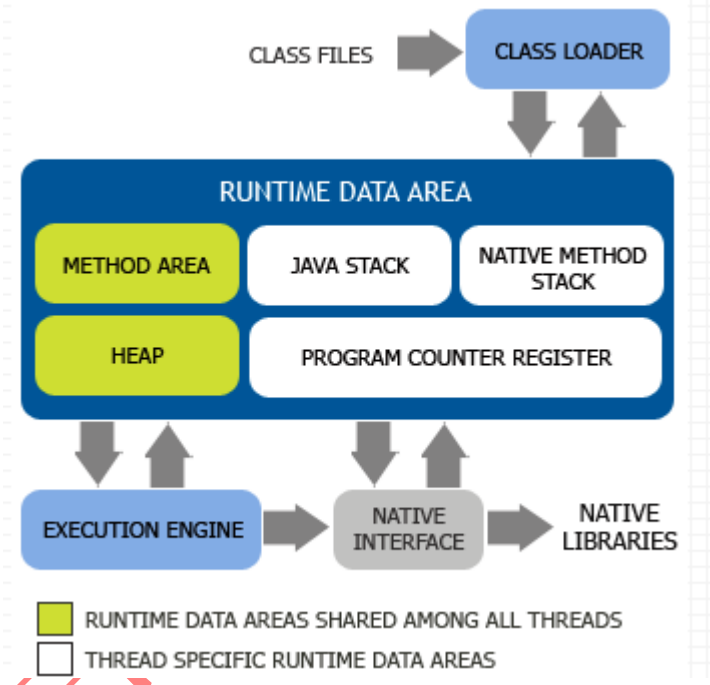
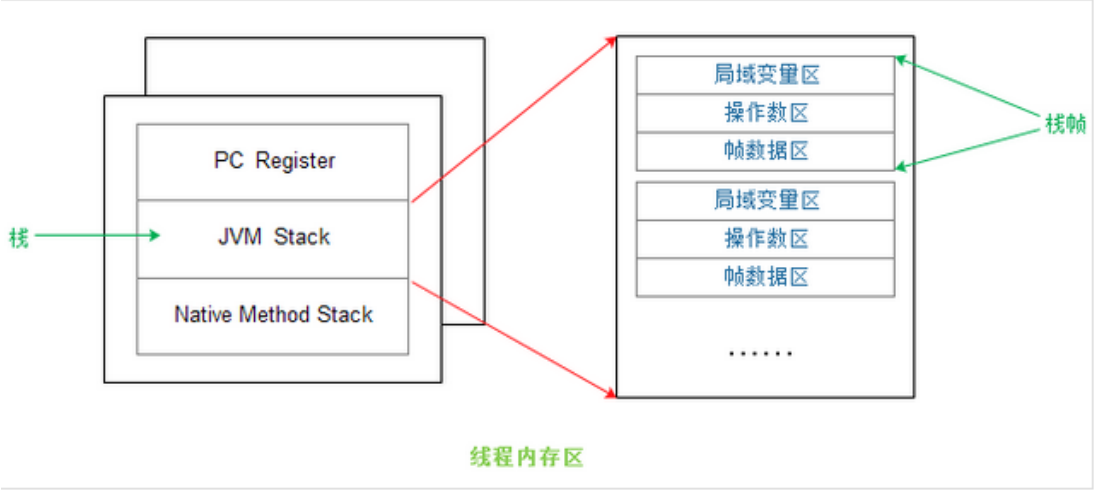
JVM 相关面试题

1. JVM 运行时内存结构

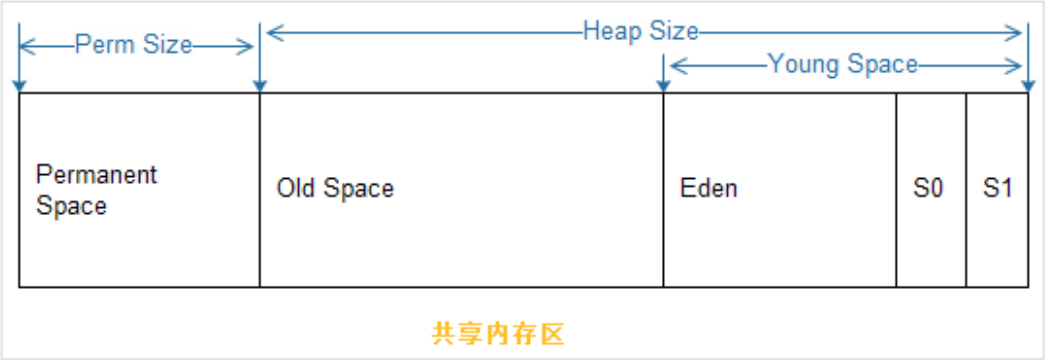
1.由如下图构成。Runtime Data Area 有如下几个区，其中 PC 程序计数器、虚拟机栈和本地方法栈是线程独享的，堆和方法区是线程共享的。Classload 用来加载 class 文件，执行引擎用来执行程序，本地方法接口调用本地方法库。



2.java stack 比较简单，每一个都是一个栈帧，每个栈帧由三部分构成。局部变量区、操作数区和帧数据区。局部变量是一个以数组形式管理的内存区，一般第 0 位是指向自己的 this 引用；其他的都是基本数据类型和 reference 类型和 returnedAddress 类型。操作数区不是通过索引来访问，通过入栈出栈来访问，是临时数据的存储区域，比方说数学计算。帧数据区是保存一些指向常量池的指针，需要常量数据时就通过这个指针来访问常量池数据。



3. 共享内存区：分为 permanent space、old space、From survivor、To survivor 和 Eden。其中 premanent 包括 runtime constant pool 和已加载的类信息和方法信息。Old space（tenured generation）包含生命周期长的存活对象。From survivor 和 Eden 存放存活比较短的对象，To survivor 是用来复制保存存活的对象。



4.JVM 参数设置。

堆：

-Xmx:最大堆内存,如：-Xmx512m

-Xms:初始时堆内存,如：-Xms256m

-XX:MaxNewSize:最大年轻区内存

-XX:NewSize:初始时年轻区内存.通常为 Xmx 的 1/3 或 1/4。新生代 = Eden + 2 个 Survivor 空间。实际可用空间为 = Eden + 1 个 Survivor，即 90%

-XX:MaxPermSize:最大持久带内存

-XX:PermSize:初始时持久带内存

-XX:+PrintGCDetails。打印 GC 信息

-XX:NewRatio 新生代与老年代的比例，如 –XX:NewRatio=2，则新生代占整个堆空间的 1/3，老年代占 2/3

-XX:SurvivorRatio 新生代中 Eden 与 Survivor 的比值。默认值为 8。即 Eden 占新生代空间的 8/10，另外两个 Survivor 各占 1/10

栈：

-xss:设置每个线程的堆栈大小. JDK1.5+ 每个线程堆栈大小为 1M，一般来说如果栈不是很深的话， 1M 是绝对够用了的。List 集合存储元素特点？

2. 对象和内存溢出

1. 对象。

A．创建。首先检查指令的参数能不能在常量区找到类的符号引用，并检查这个类是否加载、解析和初始化过，如果没有就执行类的加载过程。其次是内存分配，类加载之后就知道要分配的内存大小，分配方法有两种，一种是指针碰撞，就是一块内存是使用过的，一块是未使用的，用一个指针分割，新分配的内存指针就向空闲的挪动，compact 功能的虚拟机是用指针碰撞；另一种是空闲列表，就是一个列表记录空闲的内存块，不断更新列表，新分配的内存在列表中寻找一个合适大小的内存块，sweep 功能的虚拟机是使用空闲列表。第三，在分配内存空间的时候，还要考虑并发性。有两个方法，一种是同步处理，如采用 CAS 和失败重试的方法；另外一种是把内存分配动作按照线程划分在不同的空间之中，每个线程在堆中预先分配一小块内存，本地线程分配缓冲 TLAB，那个线程需要分配内存在那个 TLAB 上分配，只有 TLAB 用完了，才要同步锁定，重新分配。第四、对对象进行必要设置，比方说对象属于那个类，如何找到类的元数据信息和对象 hashcode 以及对象 GC 分代年龄等。

B．对象的内存布局。分为对象头、实例数据和对齐填充。对象头包括两部分，第一部分是存储对象自身信息，如 hashcode，GC 分代年龄，锁状态等；第二部分是类型指针，对象指向它的类的元数据的指针，虚拟机通过这个指针确定这是那个类的实例。

C．对象访问定位。两种方式，一种是句柄访问，句柄池有访问对象实例数据的指针和访问对象数据类型的指针。这个访问最大好处是 reference 是稳定的句柄池地址，对象改变都是改变句柄池里面的指针，而 reference 本身不动。另外

一种就是直接指针，它有到对象类型数据的指针和实例数据。这个访问的好处是速度更快，节省了一次指针定位的开销。

2. 内存溢出 OOM。

A．堆溢出。堆存放的是对象实例，只要不断创建对象，并且保证 GC Root 到对象有可大路径避免被垃圾回收清除掉对象，那么对象数量达到最大堆容量限制就会 OOM。用内存映象分析工具，Eclipse Memory Analyzer 分析一下。

B．虚拟机栈和本地方法栈溢出。分为两种，一种是如果线程请求的栈深度大于虚拟机所允许的最大深度，抛出 StackOverFlowError 异常；另一种是如果虚拟机在扩展栈时无法申请到足够内存空间，抛出 OutOfMemoryError 异常。可以减小最大堆和栈容量来获取更多的线程数量。

C．方法区和常量池溢出。会有额外提示 PermGen space。

D．本机直接内存溢出。这个 Heap Dump 文件看不到内存占用，但是如果有直接或简介使用了 NIO，那有可能就是本机直接内存溢出了。

3. GC 算法

1. 判断对象可以回收。

A．引用计数器方法。对象被引用就加一，失效的时候减一；为 0 时就可以释放。缺点是，GC 有环的时候不能释放。

B．GC Root。以根对象为起点，然后根据关联关系向下搜索。如果根对象找不到任何路径与之相连，就判断为对象可以被回收。GC Root 的选取有四种：虚拟机栈中引用对象，方法区中类的静态属性引用对象，方法区中常量引用对象和本地方法栈中 jni 引用对象。可达性分析算法标记了的对象，默认是第一次标记，如果这个对象没有实现 finalize 方法的话，就直接回收了；如果实现了 finalize 方法的话，就要把第一次标记的对象放到一个 F-Queue 队列里面，然后虚拟机会启动一个 Finalizer 线程去执行，然后会进行第二次标记。第二次标记的对象就直接回收。要想存活一次的话，就重写 finalize 方法，如果想复活的话，就在 finalize 里，把自己关联到任何一个上就行，如，把自己赋值给某个类变量或者对象的成员变量。

C．强引用对象任何时候都不会被回收；软引用在内存够的时候不会被回收，在不够时候会，SoftReference 类；弱引用在虚拟机回收时直接回收，

WeakReference。虚引用不会构成任何对对象的影响，创建目的是为了对象被回收时系统得到一个通知消息，PhantomReference。

2. 垃圾回收算法。

A．标记-清除（mark-sweep）。缺点是内存碎片多。

B．标记-复制（mark-copy）。将内存分为两块，一块内存保留对象的全部复制到另一块空闲内存中。缺点是内存减半。所以分区来实现，eden，s0，s1，按一定比例，默认 8：1：1 即可。这个就是 Minor GC，一般情况下，对象在 Eden 上申请空间，当发现没有足够空间，就发生 Minor GC，会把 Eden 和 From survivor 里的保留的对象复制到 To survivor 里面，然后清空 Eden 和 From survivor 内存，然后 From survivor 和 To survivor 对换。如果 To survivor 空间不够，直接把 From survivor 对象复制到 old space，或者部分年龄足够了的对象也会直接复制到 old space。From survivor 把对象复制到 Tenured 区域时，如果设置了 HandlePromotionFailure（允许担保失败），如果允许就只进行 Minor GC，如果不允许就触发 Full GC。

C．标记-整理（mark-compact）。整理时，先清除掉应该清除的对象，然后把存活的对象压缩到堆的一端，按顺序排放。Full GC 需要对整个堆进行回收。比方说 Tenured 满了，permanent 满了，system.gc 显示调用，会发生 Full GC。Permanent 要回收，一般回收的是常量池的常量和无用的类信息。类所有实例都回收了，加载类的 classloader 已经被回收了和类的 class 对象没有被引用（没有通过反射引用该类）这三条满足了才会回收类。

4. Class 文件结构。

A．魔数。CAFEBABE。咖啡宝贝。

B．次版本号和主版本号，1.7 是 51。

C．常量池计数器和常量池数据区。常量池表索引在 1-constant_pool_count 内的才是有效的，第一个是指向 null。

D．访问标志，access_flags.主要是 public final super interface abstract enum 等。是某个类或者接口的访问权限。后面还有有方法和变量的，类似这个，但是标志不完全一样。

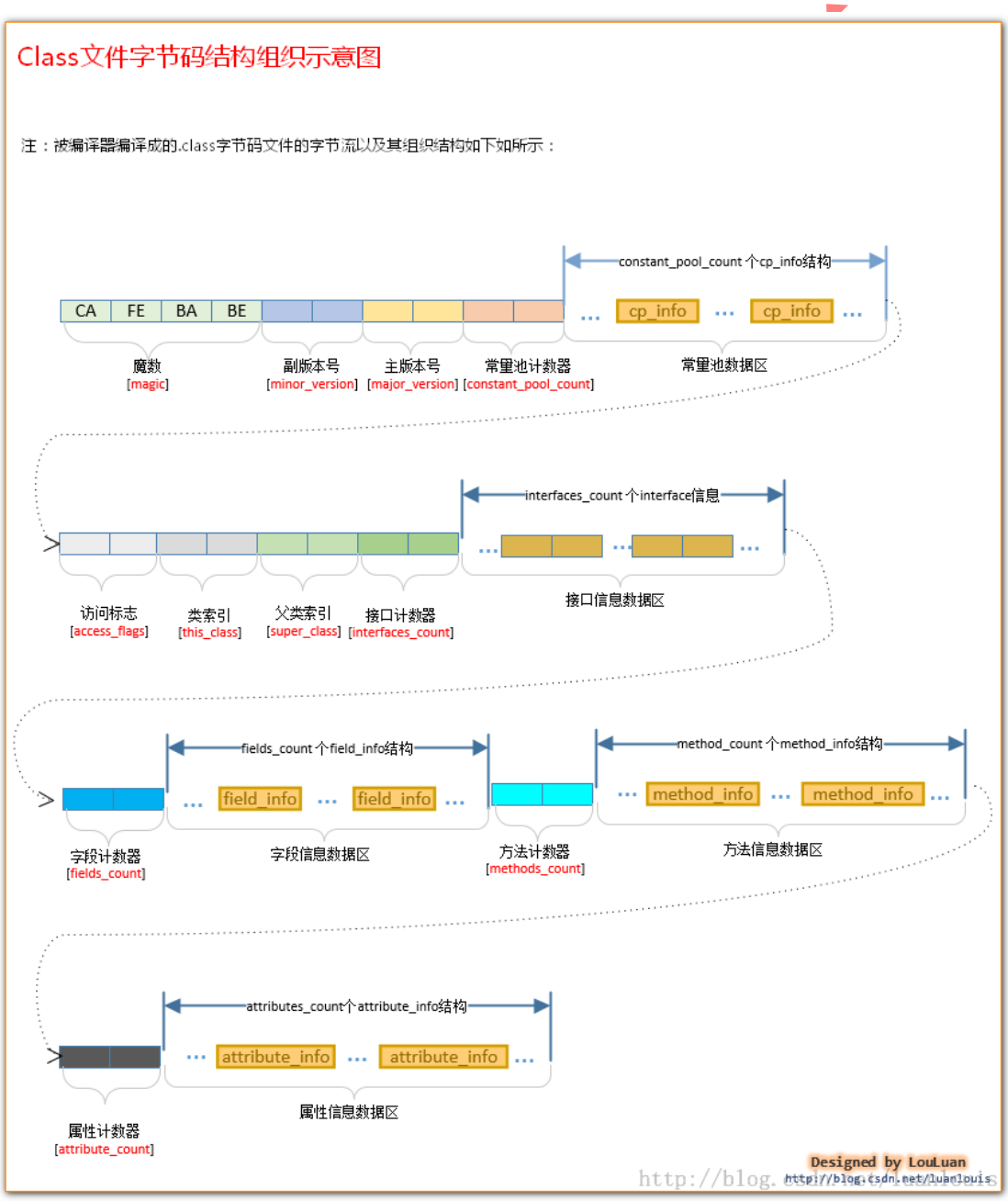
E．类索引，父类索引。this_class 和 super_class。

F．接口计数器和接口信息数据区。这里每一个实现的接口都存在接口信息数据区，从第 0 个开始到 interfaces_count。

G．字段计数器和字段信息数据区。保存的是类的所有字段信息。

H．方法计数器和方法信息数据区。保存的是类的所有方法信息。

I．属性计数器和属性信息数据区。保存属性表。常用的是 Code，Deprecated，ConstantValue，Exceptions 等。



5. 类的加载

A． 装载，查找和导入 class 文件。加载一般时机：1.如果一个类在一下三种情况下还没有初始化的话就要执行初始化：a.new 一个对象的时候，b.读取和设置一个静态字段（final 修饰的除外），c.调用一个类的静态方。2.通过 reflection 这个包调用时，如果类没有加载就要先初始化。3.一个类的子类调用的时候，其父类如果还没初始化，就要先初始化父类。4.虚拟机启动的时候，必须有一个包含 Main 方法的类先初始化。5.一个 java.lang.invoke.MethodHandle 实例最后解析结果为：REF_getStatic、REF_putStatic 和 REF_invokeStatic 的方法句柄，并且这个方法句柄所对应的类没有初始化，就要先初始化。

B．链接，分为三步。第一步是校验，检查载入的 class 文件数据的正确性，、包括文件格式、元数据验证、字节码验证和符号引用验证。第二步是准备阶段，把类变量（static）在方法区分配储存空间。第三步是解析，把符号引用转成直接引用，类和接口解析，字段和接口方法解析。

C．初始化。类的话限制性构造函数的<clinit>然后执行然后执行实例对象的<init>，当然要先执行父类的<clinit>;接口的话，不用先执行父接口的<clinit>。也就是说子类先调用父类的静态代码块，然后是子类的静态代码块。

6. 类加载器

A、 Bootstrap ClassLoader：将存放于<JAVA_HOME>\lib 目录中的，或者被-Xbootclasspath 参数所指定的路径中的，并且是虚拟机识别的（仅按照文件名识别，如 rt.jar 名字不符合的类库即使放在 lib 目录中也不会被加载）类库加载到虚拟机内存中。启动类加载器无法被 Java 程序直接引用。

B、 Extension ClassLoader：将<JAVA_HOME>\lib\ext 目录下的，或者被 java.ext.dirs 系统变量所指定的路径中的所有类库加载。开发者可以直接使用扩展类加载器。

C、 Application ClassLoader：负责加载用户类路径(ClassPath)上所指定的类库,开发者可直接使用。

D、 工作过程：如果一个类加载器接收到了类加载的请求，它首先把这个请求委托给他的父类加载器去完成，每个层次的类加载器都是如此，因此所有的加载请求都应该传送到顶层的启动类加载器中，只有当父加载器反馈自己无法完成这个加载请求（它在搜索范围中没有找到所需的类）时，子加载器才会尝试自己去加载。

E、 好处：java 类随着它的类加载器一起具备了一种带有优先级的层次关系。例如类 java.lang.Object，它存放在 rt.jar 中，无论哪个类加载器要加载这个类，最终都会委派给启动类加载器进行加载，因此 Object 类在程序的各种类加载器环境中都是同一个类。相反，如果用户自己写了一个名为 java.lang.Object 的类，并放在程序的 Classpath 中，那系统中将会出现多个不同的 Object 类，java 类型体系中最基础的行为也无法保证，应用程序也会变得一片混乱。

7. 在程序中怎么应用垃圾回收，注意那些问题

1) 尽早释放无用对象的引用，赋空值；

2) 谨慎使用集合数据类型，如数组、树、图、链表等数据结构，因为 GC 比较复杂；

3) 避免显示申请数组内存，不得不显示申请时，尽量准确估计其合理值；

4) 尽量避免在类的默认构造器中创建和初始化大量的对象，防止在调用其子类的构造器时造成不必要的资源浪费；

5) 尽量避免强制系统做垃圾回收，增长系统做垃圾回收的最终时间；

6) 尽量在合适的场景下使用对象池技术以提高系统性能；

7) 尽量做远程方法调用类应用开发时使用瞬间值变量，除非远程调用端需要获取该瞬间值变量的值。