JVM

内存结构

堆

虚拟机中最大的内存空间，所以线程共享，当一个对象被创建时，它会被分配在堆中，并返回一个指向该对象的引用

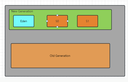
-Xms：堆的初始大小

-Xmx：堆的最大值

新生代（New Generation）

- Eden空间
- From Survivor空间
- To Survivor空间

当新对象创建时，它们会被分配到Eden空间。
当Eden空间满时，会进行Minor GC，将存活的对象移动到其中一个Survivor空间。
在下一次Minor GC时，存活的对象将被移动到另一个Survivor空间。
在对象经过多次Minor GC后，如果它还存活，就会被移动到老年代



老年代（Old Generation）

方法区

是线程共享区域

JDK1.8以前，方法区别名永久代

- -XX:PermSize：永久代的初始大小
- -XX:MaxPermSize：永久代的最大值

JDK1.8以后，方法区别名元空间

- -XX:MetaspaceSize：原空间的初始大小
- -XX:MaxMetaspaceSize：元空间的最大值

元数据

- 访问标志
- 父类
- 接口信息
- 类的字段信息
- 类的方法信息
- 构造函数信息

常量池

- 字面量
- 符号引用
  - 方法
  - 字段的引用

静态变量

即时编译器编译后的数据

虚拟机栈

每个线程在运行时都会创建一个虚拟机栈（私有线），虚拟机栈由多个栈帧组成

-Xss参数控制栈的大小

默认值

- Windows　1MB
- Linux　2MB
- Mac　2MB

栈帧

局部变量表（数组）

- 方法参数
- 方法内部定义的局部变量
- 临时变量

方法操作数栈

局部变量表的引用

操作数

数据类型

- 基本数据类型　一个槽位（slot）
- 对象的引用　一个槽位（slot）
- returnAddress类型（指向当前方法的返回地址）　两个槽位

maxstack默认为16，表示可以存储16个操作数，在字节码文件中可修改大小
Java解译器会自己计算一个值，超过默认值16时，会动态修改

动态链接　指向当前方法所属类的运行时常量池中的引用，用于支持方法的调用

方法返回地址　记录了当前方法执行完成后要返回的位置，可以是指令地址或异常处理器的地址

附加信息　虚拟机可以利用这些信息实现其他功能，例如调试、异常处理等

每个栈帧对应一个方法的调用，即方法的执行过程

本地方法栈

用于JVM执行本地方法，私有线程

-Xoss：本地方法栈大小

程序计数器

程序计数器是一块较小的内存空间，它可以看作是当前线程所执行的字节码的行号指示器，
用于记录下一条要执行的指令在字节码文件中的地址，并在线程切换后恢复执行位置

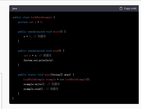
告诉机器跑哪行代码

内存模型Java Memory Model

happens-before关系向程序员提供跨线程的内存可见性保证（如果A线程的写操作a与B线程的读操作b之间存在happens-before关系，尽管a操作和b操作在不同的线程中执行，但JMM向程序员保证a操作将对b操作可见）

单线程happen-before原则：在同一个线程中，书写在前面的操作将对后面的操作可见

由于 write 方法 happens-before read 方法，所以执行结果一定是 1



锁的happen-before原则：锁定规则指的是，一个unlock操作先行发生于后面对同一个锁的lock操作

由于 write 方法和 read 方法都是在同一个对象上加锁的，所以执行结果一定是 1

volatile的happen-before原则：volatile变量规则指的是，对一个volatile变量的写操作先行发生于后面对这个变量的读操作

由于 x 是 volatile 变量，所以 write 方法 happens-before read 方法，执行结果一定是 1

happen-before的传递性原则：如果A操作 happen-before 操作，B操作happen-before C操作，那么A操作happen-before C操作

在这个示例中，MyRunnable类的run()方法将value变量进行了赋值，而getValue()方法返回value的值，在主线程中，创建了一个MyRunnable实例，并在新线程中运行它，主线程调用了t.join()方法，以确保新线程执行完毕后，主线程才能继续执行。最后，主线程调用getValue()方法获取value的值，并打印出来。
根据happens-before的传递性原则，新线程中的value赋值操作happens-before新线程的结束（在本例中是通过t.join()方法实现的），而新线程的结束又happens-before主线程中调用getValue()方法。因此，可以得出结论：新线程中的value赋值操作 happens-before 主线程中调用getValue()方法，所以getValue()方法总是返回1。
总之，当操作A happens-before 操作B，并且操作B happens-before 操作C时，操作A happens-before 操作C。这是happens-before的传递性原则。

happens-before八大原则

具体来说，如果线程A在调用线程B的start()方法后动线程B之前，对某个共享变量执行了写操作，那么在线程B中该共享变量的读操作happens-before 于线程A中对共享变量的写操作。
下面是一个简单的示例代码，其中一个线程写入共享变量，另一个线程读取共享变量。线程启动原则保证了写操作的结果对读操作可见

线程启动的happens-before原则：同一个线程的start方法happens-before此线程的其它方法

在上面的代码中，主线程对共享变量num先执行了写操作，将num的值设为2，然后启动了另一个线程。在线程中，将共享变量num的值设为1。由于线程启动原则，t中的写操作happens-before 于主线程中的写操作，所以主线程中读取共享变量num的值时会得到1，而不是2。
需要注意的是，如果在线程中对共享变量执行了写操作，那么线程启动原则并不能保证在主线程中对共享变量num的写操作happens-before于在线程中的写操作。因此，在多线程编程中，应当避免多个线程对同一个共享变量进行写操作，或者通过加锁等手段来保证同步。

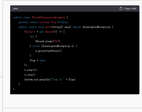
具体来说，如果线程A在某个时刻调用线程B的 interrupt() 方法，那么在线程 A 中的所有操作（happens-before 该调用）都会被视为在线程 B 中发生。
也就是说，在线程 A 中的所有写操作都会在线程 B 中的读操作之前完成，这意味着线程 A 的修改可以被线程 B 看到。

线程中断的happen-before原则：对线程interrupt方法的调用happen-before被中断线程的检测到中断发送的代码

在这个示例中，有两个线程 thread1 和 thread2，thread1 会在运行 5 秒钟后自动结束，而 thread2 会不停地输出一条消息，直到被中断为止。在主线程中，我们等待了 2 秒钟，然后中断了 thread2。
在这个示例中，我们使用了 Thread.interrupted() 方法来检查当前线程是否被中断。这个方法会清除线程的中断状态。当线程 thread2 被中断时，这个方法会返回 true，从而退出了循环。
根据 happens-before 线程中断原则，当我们在主线程中调用 thread2.interrupt() 时，它会和 thread2 中的所有语句操作建立一个 happens-before 关系。这意味着，当 thread2 中的循环检查到中断标志被设置后，它可以确定在这个标志被设置之前，所有的写操作都已经完成了。因此，Thread2 interrupted! 这个消息一定会被打印出来，而不会陷入死循环。

线程的终结包括两种情况：一种是线程正常执行完毕，另一种是线程执行过程中发生了异常而被迫终止

线程终结的happens-before原则：线程中的所有操作都happen-before线程的终止检测

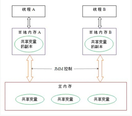


上述代码中，主线程会等待新线程t执行完毕之后才会继续往下执行，即当t线程执行完毕后，主线程才会输出flag is true。这是因为happens-before线程终结原则的作用，保证了线程内的修改对主线程的可见性。
在这个例子中，线程t的执行完后，共内部的变量flag将设置为true，按照happens-before线程终结原则，线程t的终结操作必须发生在它的所有操作之后，因此，当线程t结束时，它的所有操作必须在该线程结束之前被完全执行，而由于join方法的调用，主线程会一直等待t线程执行完毕才会继续执行，所以当主线程打印flag变量的值时，它已经被线程t修改过了，所以打印出来的结果是flag is true。

happens-before对象创建原则是指在一个线程中，如果在构造函数中给一个变量赋值，并且这个变量的引用在构造函数外部可见，那么其他线程在获取这个变量时，能够看到已经构造的对象的最新状态，不会看到一个部分构造的对象。
这条原则保证了线程之间的可见性和正确性，避免了由于对象创建不完整导致的并发问题。

对象创建的happen-before原则：一个对象的初始化完成先于他的finalize方法调用

在上面的示例代码中，有一个包含两个变量的类Example，在构造函数中，首先给变量num赋值为1，然后给变量flag赋值为true。在display方法中，如果变量flag为true，则输出变量num的值。
按照happens-before对象创建原则的要求，在一个线程中，在构造函数中给变量赋值，然后在构造函数外部使用这个变量，其他线程能够看到已经构造的对象的最新状态。
在本示例中，构造函数中首先给变量num赋值为1，然后给变量flag赋值为true。这些操作都在同一个线程内执行，因此满足happens-before对象创建原则。
在display方法中，如果变量flag为true，则输出变量num的值，其他线程也能够看到构造函数中对变量num的赋值操作，因此不会看到一个部分构造的对象。



图例

可见性关键字

- volatile　保证可见性和有序性
- synchronized　保证可见性和有序性；通过管程（Monitor）保证一组动作的原子性
  不保证同步块内的代码整点重排序，因为它通过锁保证同一时刻只有一个线程访问同步块（或临界区），
  也能是通同步块内的代码只能满足 as-if-serial 语义，只是单线程的执行结果不改变，可以进行重排序。
- final　通过禁止在构造函数初始化和给 final 字段赋值这两个动作的重排序，保证可见性（如果 this 引用逃逸就不好说可见性了）

虚拟机子系统

执行引擎

实现

- 即时编译器　将字节码转为机器指令，提高执行效率
- 字节码解释器　将字节码解释为可执行的指令序列

字节码解释器解释字节码，即时编译器优化热点代码，
在Java程序的运行过程中，二者交替工作，共同提高了程序的执行效率

组成部分

- PC寄存器
- 方法区和堆
- 操作数栈和局部变量表
- 异常处理器

类加载器

将Java类加载到虚拟机中，并转化为字节码　https://www.processon.com/mindmap/641712d033b64141c5cc7a2d

垃圾回收器

回收Java中不在使用的内存空间　https://www.processon.com/mindmap/642a92a1bb6814436aa723f78