## 为什么需要线程 + 面试题

### 线程介绍

线程（Thread）是程序运行的执行单元，依托于进程存在。一个进程中可以包含多个线程，多线程可以共享一块内存空间和一组系统资源，因此线程之间的切换更加节省资源、更加轻量化，因而也被称为轻量级的进程。

#### 什么是进程

进程（Processes）是程序的一次动态执行，是系统进行资源分配和调度的基本单位，是操作系统运行的基础，通常每一个进程都拥有自己独立的内存空间和系统资源。简单来说，进程可以被当做是一个正在运行的程序。

#### 为什么需要线程

程序的运行必须依靠进程，进程的实际执行单元就是线程。

#### 为什么需要多线程

多线程可以提高程序的执行性能。例如，有个 90 平方的房子，一个人打扫需要花费 30 分钟，三个人打扫就只需要 10 分钟，这三个人就是程序中的“多线程”。

### 线程使用

线程的创建，分为以下三种方式：

* 继承 Thread 类，重写 run 方法
* 实现 Runnable 接口，实现 run 方法
* 实现 Callable 接口，实现 call 方法

下面分别来看看线程创建和使用的具体代码。

#### 1）继承 Thread 类

请参考以下代码：

class ThreadTest {  
 public static void main(String[] args) throws Exception {  
 MyThread thread = new MyThread();  
 thread.start();  
 }  
}  
class MyThread extends Thread {  
 @Override  
 public void run() {  
 System.out.println("Thread");  
 }  
}

以上程序执行结果如下：

Thread

#### 2）实现 Runnable 接口

请参考以下代码：

class ThreadTest {  
 public static void main(String[] args) {  
 MyRunnable runnable = new MyRunnable();  
 new Thread(runnable).start();  
 }  
}  
class MyRunnable implements Runnable {  
 @Override  
 public void run() {  
 System.out.println("Runnable");  
 }  
}

以上程序执行结果如下：

Runnable

#### 3）实现 Callable 接口

请参考以下代码：

class ThreadTest {  
 public static void main(String[] args) throws Exception {  
 MyCallable callable = new MyCallable();  
 // 定义返回结果  
 FutureTask<String> result = new FutureTask(callable);  
 // 执行程序  
 new Thread(result).start();  
 // 输出返回结果  
 System.out.println(result.get());  
 }  
}  
class MyCallable implements Callable {  
 @Override  
 public String call() {  
 System.out.println("Callable");  
 return "Success";  
 }  
}

以上程序执行结果如下：

Callable

Success

可以看出，Callable 的调用是可以有返回值的，它弥补了之前调用线程没有返回值的情况，它是随着 JDK 1.5 一起发布的。

#### 4）JDK 8 创建线程

JDK 8 之后可以使用 Lambda 表达式很方便地创建线程，请参考以下代码：

new Thread(() -> System.out.println("Lambda Of Thread.")).start();

### 线程高级用法

##### 线程等待

使用 wait() 方法实现线程等待，代码如下：

System.out.println(LocalDateTime.now());  
Object lock = new Object();  
Thread thread = new Thread(() -> {  
 synchronized (lock){  
 try {  
 // 1 秒钟之后自动唤醒  
 lock.wait(1000);  
 System.out.println(LocalDateTime.now());  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
});  
thread.start();

以上程序执行结果如下：

2019-06-22T20:53:08.776

2019-06-22T20:53:09.788

注意：当使用 wait() 方法时，必须先持有当前对象的锁，否则会抛出异常 java.lang.IllegalMonitorStateException。

##### 线程唤醒

使用 notify()/notifyAll() 方法唤醒线程。

* notify() 方法随机唤醒对象的等待池中的一个线程；
* notifyAll() 唤醒对象的等待池中的所有线程。

使用如下：

Object lock = new Object();  
lock.wait();  
lock.notify();  
// lock.notifyAll();

##### 线程休眠

// 休眠 1 秒  
Thread.sleep(1000);

##### 等待线程执行完成

Thread joinThread = new Thread(() -> {  
 try {  
 System.out.println("执行前");  
 Thread.sleep(1000);  
 System.out.println("执行后");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
});  
joinThread.start();  
joinThread.join();  
System.out.println("主程序");

以上程序执行结果：

执行前

执行后

主程序

##### yield 交出 CPU 执行权

new Thread(){  
 @Override  
 public void run() {  
 for (int i = 1; i < 10; i++) {  
 if (i == 5) {  
 // 让同优先级的线程有执行的机会  
 this.yield();  
 }  
 }  
 }  
}.start();

注意：yield 方法是让同优先级的线程有执行的机会，但不能保证自己会从正在运行的状态迅速转换到可运行的状态。

##### 线程中断

使用 System.exit(0) 可以让整个程序退出；要中断单个线程，可配合 interrupt() 对线程进行“中断”。  
使用代码如下：

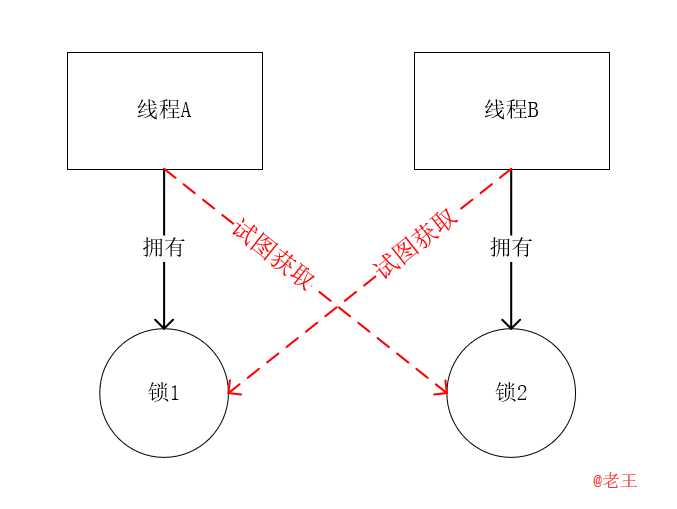
Thread interruptThread = new Thread() {  
 @Override  
 public void run() {  
 for (int i = 0; i < Integer.MAX\_VALUE; i++) {  
 System.out.println("i：" + i);  
 if (this.isInterrupted()) {  
 break;  
 }  
 }  
 }  
};  
interruptThread.start();  
Thread.sleep(10);  
interruptThread.interrupt();

##### 线程优先级

在 Java 语言中，每一个线程有一个优先级，默认情况下，一个线程继承它父类的优先级。可以使用 setPriority 方法设置（1-10）优先级，默认的优先级是 5，数字越大表示优先级越高，优先级越高的线程可能优先被执行的概率就越大。  
设置优先级的代码如下：

Thread thread = new Thread(() -> System.out.println("Java"));  
thread.setPriority(10);  
thread.start();

### 死锁

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。  
比如，当线程 A 持有独占锁 a，并尝试去获取独占锁 b 的同时，线程 B 持有独占锁 b，并尝试获取独占锁 a 的情况下，就会发生 A B 两个线程由于互相持有对方需要的锁，而发生的阻塞现象，我们称为死锁。  
死锁示意图如下所示：  
  
死锁代码：

Object obj1 = new Object();  
Object obj2 = new Object();  
// 线程1拥有对象1，想要等待获取对象2  
new Thread() {  
 @Override  
 public void run() {  
 synchronized (obj1) {  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 synchronized (obj2) {  
 System.out.println(Thread.currentThread().getName());  
 }  
 }  
 }  
}.start();  
// 线程2拥有对象2，想要等待获取对象1  
new Thread() {  
 @Override  
 public void run() {  
 synchronized (obj2) {  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 synchronized (obj1) {  
 System.out.println(Thread.currentThread().getName());  
 }  
 }  
 }  
}.start();

### 相关面试题

#### 1.线程和进程有什么区别和联系？

答：从本质上来说，线程是进程的实际执行单元，一个程序至少有一个进程，一个进程至少有一个线程，它们的区别主要体现在以下几个方面：

* 进程间是独立的，不能共享内存空间和上下文，而线程可以；
* 进程是程序的一次执行，线程是进程中执行的一段程序片段；
* 线程占用的资源比进程少。

#### 2.如何保证一个线程执行完再执行第二个线程？

答：使用 join() 方法，等待上一个线程的执行完之后，再执行当前线程。  
示例代码：

Thread joinThread = new Thread(() -> {  
 try {  
 System.out.println("执行前");  
 Thread.sleep(1000);  
 System.out.println("执行后");  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
});  
joinThread.start();  
joinThread.join();  
System.out.println("主程序");

#### 3.线程有哪些常用的方法？

答：线程的常用方法如下：

* currentThread()：返回当前正在执行的线程引用
* getName()：返回此线程的名称
* setPriority()/getPriority()：设置和返回此线程的优先级
* isAlive()：检测此线程是否处于活动状态，活动状态指的是程序处于正在运行或准备运行的状态
* sleep()：使线程休眠
* join()：等待线程执行完成
* yield()：让同优先级的线程有执行的机会，但不能保证自己会从正在运行的状态迅速转换到可运行的状态
* interrupted()：是线程处于中断的状态，但不能真正中断线程

#### 4.wait() 和 sleep() 有什么区别？

答：wait() 和 sleep() 的区别主要体现在以下三个方面。

* 存在类的不同：sleep() 来自 Thread，wait() 来自 Object。
* 释放锁：sleep() 不释放锁；wait() 释放锁。
* 用法不同：sleep() 时间到会自动恢复；wait() 可以使用 notify()/notifyAll() 直接唤醒。

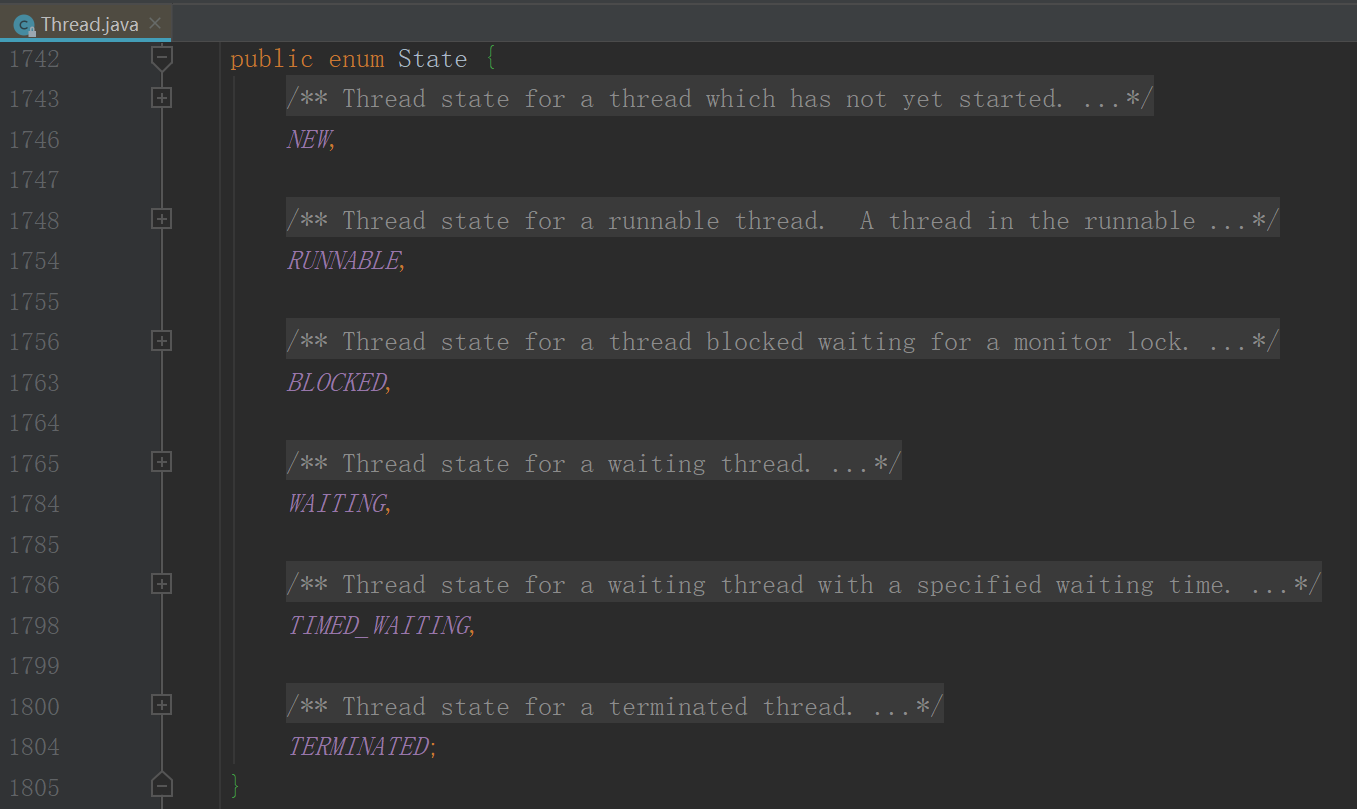
#### 5.守护线程是什么？

答：守护线程是一种比较低级别的线程，一般用于为其他类别线程提供服务，因此当其他线程都退出时，它也就没有存在的必要了。例如，JVM（Java 虚拟机）中的垃圾回收线程。

#### 6.线程有哪些状态？

答：在 JDK 8 中，线程的状态有以下六种。

* NEW：尚未启动
* RUNNABLE：正在执行中
* BLOCKED：阻塞（被同步锁或者 IO 锁阻塞）
* WAITING：永久等待状态
* TIMED\_WAITING：等待指定的时间重新被唤醒的状态
* TERMINATED：执行完成

题目分析：JDK 8 线程状态的源码如下图所示：  


#### 7.线程中的 start() 和 run() 有那些区别？

答：start() 方法用于启动线程，run() 方法用于执行线程的运行时代码。run() 可以重复调用，而 start() 只能调用一次。

#### 8.产生死锁需要具备哪些条件？

答：产生死锁的四个必要条件：

* 互斥条件：一个资源每次只能被一个线程使用；
* 请求与保持条件：一个线程因请求资源而阻塞时，对已获得的资源保持不放；
* 不剥夺条件：线程已获得的资源，在末使用完之前，不能强行剥夺；
* 循环等待条件：若干线程之间形成一种头尾相接的循环等待资源关系；

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

#### 9.如何预防死锁？

答：预防死锁的方法如下：

* 尽量使用 tryLock(long timeout, TimeUnit unit) 的方法 (ReentrantLock、ReentrantReadWriteLock)，设置超时时间，超时可以退出防止死锁；
* 尽量使用 Java. util. concurrent 并发类代替自己手写锁；
* 尽量降低锁的使用粒度，尽量不要几个功能用同一把锁；
* 尽量减少同步的代码块。

#### 10.thread.wait() 和 thread.wait(0) 有什么区别？代表什么含义？

答：thread.wait() 和 thread.wait(0) 是相同的，使用 thread.wait() 内部其实是调用的 thread.wait(0)，源码如下：

public final void wait() throws InterruptedException {  
 wait(0);  
}

wait() 表示进入等待状态，释放当前的锁让出 CPU 资源，并且只能等程序执行 notify()/notifyAll() 方法才会被重写唤醒。

#### 11.如何让两个程序依次输出 11/22/33 等数字，请写出实现代码？

答：使用思路是在每个线程输出信息之后，让当前线程等待一会再执行下一次操作，具体实现代码如下：

new Thread(() -> {  
 for (int i = 1; i < 4; i++) {  
 System.out.println("线程一：" + i);  
 try {  
 Thread.sleep(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}).start();  
new Thread(() -> {  
 for (int i = 1; i < 4; i++) {  
 System.out.println("线程二：" + i);  
 try {  
 Thread.sleep(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}).start();

程序执行结果如下：

线程一：1

线程二：1

线程二：2

线程一：2

线程二：3

线程一：3

#### 12.说一下线程的调度策略？

答：线程调度器选择优先级最高的线程运行，但是如果发生以下情况，就会终止线程的运行：

* 线程体中调用了 yield() 方法，让出了对 CPU 的占用权；
* 线程体中调用了 sleep() 方法，使线程进入睡眠状态；
* 线程由于 I/O 操作而受阻塞；
* 另一个更高优先级的线程出现；
* 在支持时间片的系统中，该线程的时间片用完。

### 总结

程序的运行依靠的是进程，而进程的执行依靠的是多个线程，多线程之间可以共享一块内存和一组系统资源，而多进程间通常是相互独立的。线程的创建有三种方式：继承 Thread 重写 run 方法，实现 Runnable 或 Callable 接口，其中 Callable 可以允许线程的执行有返回值，JDK 8 中也可以使用 Lambda 来更加方便的使用线程，线程是有优先级的，优先级从 1-10 ，数字越大优先级越高，也越早被执行。如果两个线程各自拥有一把锁的同时，又同时等待获取对方的锁，就会造成死锁。可以降低锁的粒度或减少同步代码块的范围或使用 Java 提供的安全类，来防止死锁的产生。

[点击此处下载本文源码](https://github.com/vipstone/java-%20interview/tree/master/interview-code/src/main/java/com/interview)

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持