线程基础

2024年4月3日 10:34

线程和进程的区别?

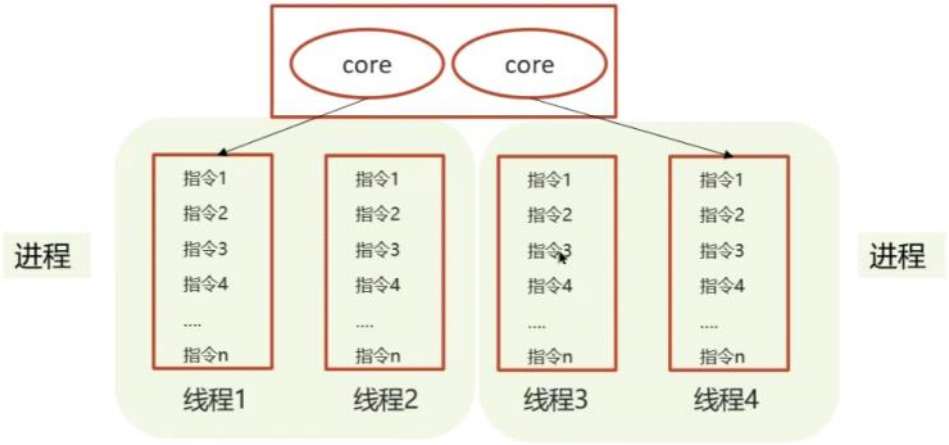
程序由指令和数据组成，但这些指令要运行，数据要读写，就必须将指令加载至 CPU，数据加载至内存。在指令运行过程中还需要用到磁盘、网络等设备。进程就是用来加载指令、管理内存、管理 IO 的。

当一个程序被运行，从磁盘加载这个程序的代码至内存，这时就开启了一个进程。

线程和进程的区别?

一个线程就是一个指令流，将指令流中的一条条指令以一定的顺序交给 CPU 执行

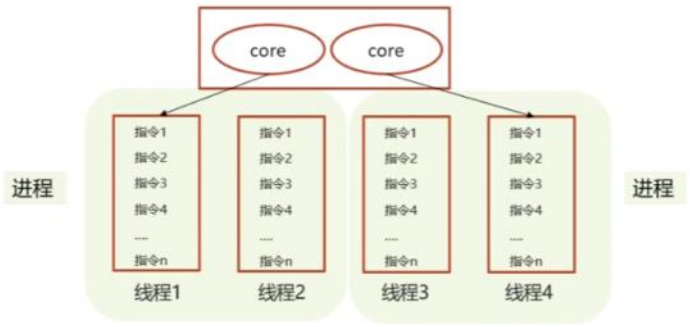
一个进程之内可以分为一到多个线程。



线程和进程的区别?

二者对比

- 进程是正在运行程序的实例，进程中包含了线程，每个线程执行不同的任务
- 不同的进程使用不同的内存空间，在当前进程下的所有线程可以共享内存空间
- 线程更轻量，线程上下文切换成本一般上要比进程上下文切换低(上下文切换指的是从一个线程切换到另一个线程)



并行和并发有什么区别?

单核CPU

- 单核CPU下线程实际还是串行执行的
- 操作系统中有一个组件叫做任务调度器，将cpu的时间片（windows下时间片最小约为 15 毫秒）分给不同的程序使用，只是由于cpu在线程间（时间片很短）的切换非常快，人类感觉是同时运行的 。
- 总结为一句话就是： 微观串行，宏观并行
- 一般会将这种线程轮流使用CPU的做法称为并发（concurrent）

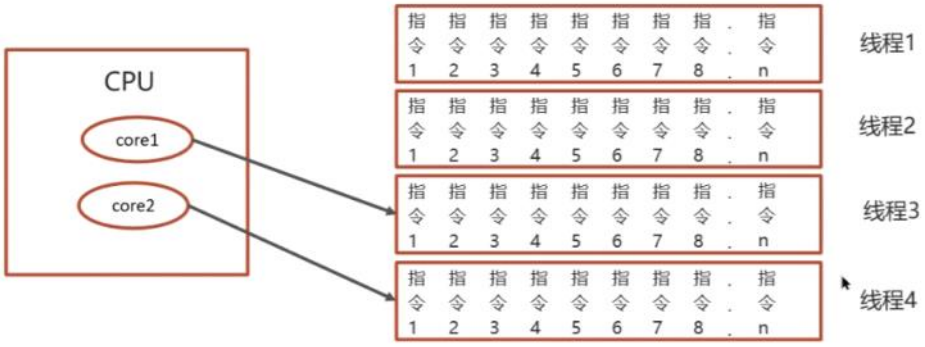
CPU	时间片1	时间片2	时间片3
core	线程1	线程2	线程3

并行和并发有什么区别?

多核CPU

每个核（core）都可以调度运行线程，这时候线程可以是并行的。

CPU	时间片1	时间片2	时间片3	时间片4
core1	线程1	线程1	线程3	线程3
core2	线程2	线程4	线程2	线程4



并行和并发有什么区别?

并发（concurrent）是同一时间应对（dealing with）多件事情的能力

并行（parallel）是同一时间动手做（doing）多件事情的能力

- 家庭主妇做饭、打扫卫生、给孩子喂奶，她一个人轮流交替做这多件事，这时就是并发
- 家庭主妇雇了个保姆，她们一起这些事，这时既有并发，也有并行（这时会产生竞争，例如锅只有一口，一个人用锅时，另一个人就得等待）
- 雇了3个保姆，一个专做饭、一个专打扫卫生、一个专喂奶，互不干扰，这时是并行

并行和并发有什么区别

现在都是多核CPU，在多核CPU下

- 并发是同一时间应对多件事情的能力，多个线程轮流使用一个或多个CPU
- 并行是同一时间动手做多件事情的能力，4核CPU同时执行4个线程

创建线程的方式有哪些?

共有四种方式可以创建线程，分别是：

- 继承Thread类
- 实现runnable接口
- 实现Callable接口
- 线程池创建线程

继承Thread类

```
public class MyThread extends Thread {

    @Override
    public void run() {
        System.out.println("MyThread...run...");
    }


    public static void main(String[] args) {

        // 创建MyThread对象
        MyThread t1 = new MyThread() ;
        MyThread t2 = new MyThread() ;

        // 调用start方法启动线程
        t1.start();
        t2.start();

    }

}
```

实现runnable接口

```
public class MyRunnable implements Runnable {

    @Override
    public void run() {
        System.out.println("MyRunnable...run...");
    }

    public static void main(String[] args) {

        // 创建MyRunnable对象
        MyRunnable mr = new MyRunnable();

        // 创建Thread对象
        Thread t1 = new Thread(mr);
        Thread t2 = new Thread(mr);

        // 调用start方法启动线程
        t1.start();
        t2.start();

    }
}
```

实现Callable接口

```
public class MyCallable implements Callable<String> {

    @Override
    public String call() throws Exception {
        System.out.println(Thread.currentThread().getName());
        return "ok";
    }

    public static void main(String[] args) throws ExecutionException, InterruptedException {
        // 创建MyCallable对象
        MyCallable mc = new MyCallable() ;
        // 创建FutureTask
        FutureTask<String> ft = new FutureTask<String>(mc);
        // 创建Thread对象
        Thread t1 = new Thread(ft) ;
        Thread t2 = new Thread(ft) ;
        // 调用start方法启动线程
        t1.start();
        // 调用ft的get方法获取执行结果
        String result = ft.get();
        // 输出
        System.out.println(result);
    }
}
```

线程池创建线程

```
public class MyExecutors implements Runnable{

    @Override
    public void run() {
        System.out.println("MyRunnable...run...");
    }

    public static void main(String[] args) {

        // 创建线程池对象
        ExecutorService threadPool = Executors.newFixedThreadPool(3);
        threadPool.submit(new MyExecutors()) ;

        // 关闭线程池
        threadPool.shutdown();

    }

}
```

runnable 和 callable 有什么区别?

参考回答:

1. Runnable 接口run方法没有返回值
2. Callable接口call方法有返回值，是个泛型，和Future、FutureTask配合可以用来获取异步执行的结果
3. Callable接口的call()方法允许抛出异常；而Runnable接口的run()方法的异常只能在内部消化，不能继续上抛

面试官再追问：在启动线程的时候，可以使用run方法吗？run()和 start()有什么区别?

线程的 run()和 start()有什么区别?

start(): 用来启动线程，通过该线程调用run方法执行run方法中所定义的逻辑代码。start方法只能被调用一次。

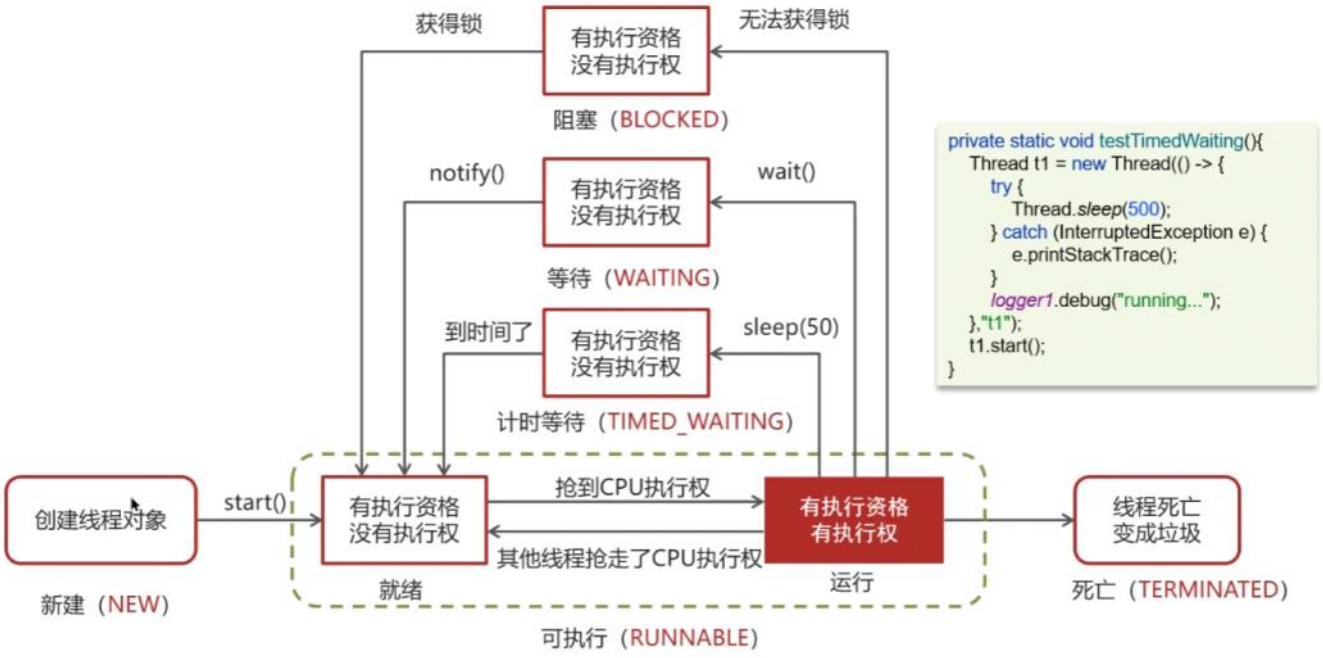
run(): 封装了要被线程执行的代码，可以被调用多次。

1. 创建线程的方式有哪些?
   - 继承Thread类
   - 实现runnable接口
   - 实现Callable接口
   - 线程池创建线程(项目中使用方式)
2. runnable 和 callable 有什么区别
   - Runnable 接口run方法没有返回值
   - Callable接口call方法有返回值，需要FutureTask获取结果
   - Callable接口的call()方法允许抛出异常；而Runnable接口的run()方法的异常只能在内部消化，不能继续上抛
3. run()和 start()有什么区别?
   - start(): 用来启动线程，通过该线程调用run方法执行run方法中所定义的逻辑代码。start方法只能被调用一次。
   - run(): 封装了要被线程执行的代码，可以被调用多次。

线程包括哪些状态，状态之间是如何变化的

线程的状态可以参考JDK中的Thread类中的枚举State

```
public enum State {

    //尚未启动的线程的线程状态
    NEW,

    //可运行线程的线程状态。
    RUNNABLE,

    //线程阻塞等待监视器锁的线程状态。
    BLOCKED,

    //等待线程的线程状态
    WAITING,

    //具有指定等待时间的等待线程的线程状态
    TIMED_WAITING,

    //已终止线程的线程状态。线程已完成执行
    TERMINATED;
}
```




```
private static void testTimedWaiting(){
    Thread t1 = new Thread(() -> {
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        logger1.debug("running...");
    },"t1");
    t1.start();
}
```

1. 线程包括哪些状态

新建（NEW）、可运行（RUNNABLE）、阻塞（BLOCKED）、等待（WAITING）、时间等待（TIMED_WALTING）、终止（TERMINATED）

2. 线程状态之间是如何变化的

- 创建线程对象是新建状态
- 调用了start()方法转变为可执行状态
- 线程获取到了CPU的执行权，执行结束是终止状态
- 在可执行状态的过程中，如果没有获取CPU的执行权，可能会切换其他状态
  - 如果没有获取锁（synchronized或lock）进入阻塞状态，获得锁再切换为可执行状态
  - 如果线程调用了wait()方法进入等待状态，其他线程调用notify()唤醒后可切换为可执行状态
  - 如果线程调用了sleep(50)方法，进入计时等待状态，到时间后可切换为可执行状态

新建 T1、T2、T3 三个线程，如何保证它们按顺序执行?

可以使用线程中的join方法解决

join()　等待线程运行结束

小例子：

t.join()

阻塞调用此方法的线程进入timed_waiting

直到线程t执行完成后，此线程再继续执行

```
Thread t1 = new Thread(() -> {
    System.out.println("t1");
}) ;
Thread t2 = new Thread(() -> {
    try {
        t1.join();                    // 加入线程t1,只有t1线程执行完毕以后，再次执行该线程
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("t2");
}) ;
Thread t3 = new Thread(() -> {
    try {
        t2.join();                    // 加入线程t2,只有t2线程执行完毕以后，再次执行该线程
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("t3");
}) ;
// 启动线程
t1.start();
t2.start();
t3.start();
```

notify()和 notifyAll()有什么区别?

- notifyAll：唤醒所有wait的线程
- notify：只随机唤醒一个 wait 线程

在java中wait和sleep方法的不同?

共同点

wait()，wait(long) 和 sleep(long) 的效果都是让当前线程暂时放弃 CPU 的使用权，进入阻塞状态

不同点

1.方法归属不同

- sleep(long) 是 Thread 的静态方法
- 而 wait()，wait(long) 都是 Object 的成员方法，每个对象都有

2.醒来时机不同

- 执行 sleep(long) 和 wait(long) 的线程都会在等待相应毫秒后醒来
- wait(long) 和 wait() 还可以被 notify 唤醒，wait() 如果不唤醒就一直等下去
- 它们都可以被打断唤醒

3. 锁特性不同（重点）

- wait 方法的调用必须先获取 wait 对象的锁，而 sleep 则无此限制
- wait 方法执行后会释放对象锁，允许其它线程获得该对象锁（我放弃 cpu，但你们还可以用）
- 而 sleep 如果在 synchronized 代码块中执行，并不会释放对象锁（我放弃 cpu，你们也用不了）

如何停止一个正在运行的线程?

有三种方式可以停止线程

- 使用退出标志，使线程正常退出，也就是当run方法完成后线程终止
- 使用stop方法强行终止（不推荐，方法已作废）
- 使用interrupt方法中断线程
  - 打断阻塞的线程（ sleep，wait，join ）的线程，线程会抛出InterruptedException异常
  - 打断正常的线程，可以根据打断状态来标记是否退出线程