线程八大核心基础

创建线程常见面试问题

1、有多少种实现多线程的方式?

```
1、从不同的角度看，会有不同的答案
2、典型答案是两种，分别是实现Runnable接口和继承Thread类
3、从原理上看，发现Thread类也实现了Runnable接口，并查看Thread类的run()方法，发现其实两者本质是一样的，run方法代码如下：
```

```
    @Override
    public void run() {
        if (target != null) {
            target.run();
        }
    }
```

```
方法一和方法二,即"继承Thread类然后重写run()方法和实现Runnable接口并实现run()方法，在实现多线程的本质上是没有区别的，最终都
这两个方法最大的区别在于两个run()方法的内容来源：
方法一：最终调用target.run();
方法二：run()方法整个被重写了
4、具体展开说其他方式
    还有其他实现线程的方法，例如线程池等，他们也能新建线程，但细看源码，也没有逃离本质，也就是实现Runnable接口和继承Thread
5、结论
    我们只能通过新建Thread类这一种方式来创建线程，但类里面的run()方法有两种方式来实现，第一种是继承Thread类并重写run()方法
    第二种是实现Runnable接口并实现run()方法，并将Runnable实例传给Thread类。
    除此之外，从表面上看，线程池、定时器等工具类也可以创建线程，但他们的本质也逃离不出刚才所说的范围。
```

启动线程常见面试问题

1、一个线程两次调用start()方法会出现什么情况？为什么?

1、既然start()方法会调用run()方法，为什么选择调用start()方法，而不是直接调用run()方法呢?

停止线程常见面试问题

1、如何停止线程?

```
1、用interrupt来请求线程停止而不是强制，好处是安全。
2、想停止线程，需要停止方、被停止方、子方法调用方相互配合才行：
    a)请求方：发出中断信号
    b)被停止方：每次循环或者适时的检查中断信号，并且在抛出InterruptedEXception地方处理该中断信号；
    c)子方法调用方（被线程调用的方法）：优先抛出InterruptedEXception，或者检测到中断信号时，再次设置中断信号；
3、最后再说错误停止线程的方法：stop()、suspend()已废弃，volatile的boolean无法处理长时间阻塞的情况。
```

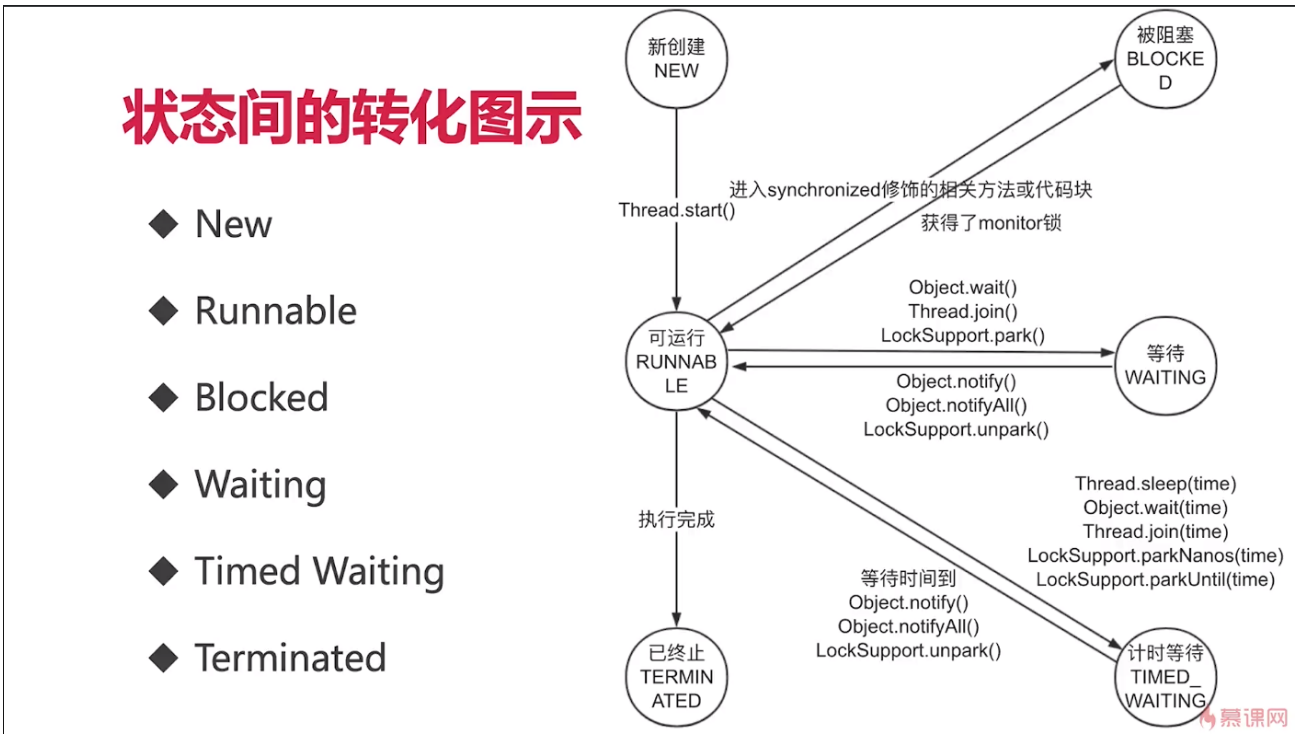
2、如何处理不可中断的阻塞?

```
具体问题具体分析，尽量使用可以响应中断的方法。
```

线程生命周期常见面试问题

1、线程有几种状态？生命周期是什么?

先讲6个圈内的状态名，再讲转换路径（例如：New只能跳转到Runnable），最后将转移条件。



2、阻塞状态的定义?

一般而言，把Blocked(被阻塞)、Waiting(等待)、Timed_waiting(计时等待)都成为阻塞状态
不仅仅是Blocked状态

Thread和Object类常见面试问题

1、wait和notify的基本用法代码演示

1. 研究代码执行顺序
2. 证明wait释放锁

```
public class Wait {

    public static Object object = new Object();

    static class Thread1 extends Thread {

        @Override
        public void run() {
            synchronized (object) {
                System.out.println(Thread.currentThread().getName() + "开始执行了");
                try {
                    object.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("线程" + Thread.currentThread().getName() + "获取到了锁。");
            }
        }
    }

    static class Thread2 extends Thread {

        @Override
        public void run() {
            synchronized (object) {
                object.notify();
```

```
                System.out.println("线程" + Thread.currentThread().getName() + "调用了notify()");
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Thread1 thread1 = new Thread1();
        Thread2 thread2 = new Thread2();
        thread1.start();
        Thread.sleep(200);
        thread2.start();
    }
}
```

2、两个线程交替打印0~100的奇偶数，用synchronized关键字实现

```
public class WaitNotifyPrintOddEvenSyn {

    private static int count;

    private static final Object lock = new Object();

    //新建2个线程
    //1个只处理偶数，第二个只处理奇数（用位运算）
    //用synchronized来通信
    public static void main(String[] args) {
        new Thread(() -> {
            while (count < 100) {
                synchronized (lock) {
                    if ((count & 1) == 0) {
                        System.out.println(Thread.currentThread().getName() + ":" + count++);
                    }
                }
            }
        }, "偶数").start();

        new Thread(() -> {
            while (count < 100) {
                synchronized (lock) {
                    if ((count & 1) == 1) {
                        System.out.println(Thread.currentThread().getName() + ":" + count++);
                    }
                }
            }
        }, "奇数").start();
    }
}
```

3、两个线程交替打印0~100的奇偶数，用wait和notify实现

```
public class WaitNotifyPrintOddEveWait {

    private static int count = 0;
    private static final Object lock = new Object();


    public static void main(String[] args) {
        new Thread(new TurningRunner(), "偶数").start();
        new Thread(new TurningRunner(), "奇数").start();
    }

    //1. 拿到锁，我们就打印
    //2. 打印完，唤醒其他线程，自己就休眠
    static class TurningRunner implements Runnable {

        @Override
        public void run() {
            while (count <= 100) {
                synchronized (lock) {
                    //拿到锁就打印
                    System.out.println(Thread.currentThread().getName() + ":" + count++);
```

```
                    lock.notify();
                    if (count <= 100) {
                        try {
                            //如果任务还没结束，就让出当前的锁，并休眠
                            lock.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        }
    }
}
```

4、手写生产者消费者模式,使wait/notify来实现

```
public class ProducerConsumerModel {
    public static void main(String[] args) {
        EventStorage eventStorage = new EventStorage();
        Producer producer = new Producer(eventStorage);
        Consumer consumer = new Consumer(eventStorage);
        new Thread(producer).start();
        new Thread(consumer).start();
    }
}

class Producer implements Runnable {

    private EventStorage storage;

    public Producer(
            EventStorage storage) {
        this.storage = storage;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            storage.put();
        }
    }
}

class Consumer implements Runnable {

    private EventStorage storage;

    public Consumer(
            EventStorage storage) {
        this.storage = storage;
    }

    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            storage.take();
        }
    }
}

class EventStorage {

    private int maxSize;
    private LinkedList<Date> storage;

    public EventStorage() {
        maxSize = 10;
        storage = new LinkedList<>();
    }

    public synchronized void put() {
        while (storage.size() == maxSize) {
```

```
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        storage.add(new Date());
        System.out.println("仓库里有了" + storage.size() + "个产品。");
        notify();
    }

    public synchronized void take() {
        while (storage.size() == 0) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("拿到了" + storage.poll() + ", 现在仓库还剩下" + storage.size());
        notify();
    }
}
```

5、为什么wait()方法需要在同步代码块内使用，而sleep不需要?

6、为什么线程间通信的方法wait()、notify()、notifyAll()被定义在Object类里？而sleep定义在Thread类里?

7、wait()方法属于Object对象，如果使用Thread.wait()方法会怎样?

线程异常处理常见面试问题

利用UncaughtExceptionHandler处理异常

1、Java异常体系

2、如何处理全局异常？为什么要全局处理？不处理行不行?

3、run()方法是否可以抛出异常？如果抛出异常，线程的状态会怎样?

4、线程中如何处理某个未处理异常?