多线程

进程: 程序的基本执行实体. **线程**: 操作系统可进行运算调度的最小单位

应用场景: 软件中的耗时操作　　迁移大文件　加载大量资源文件

让多个事情同时运行,提高效率

并发和并行

并发:同一时刻,多个指令在单个CPU上交替进行

并行:同一时刻, 多个指令在多个CPU上同时进行

实现方式

1.继承Thread类

2.实现Runnable接口

3.利用Callable和Future接口的方式实现

黑马程序员 www.itheima.com
多一句没有，少一句不行，用更短时间，教会更实用的技术！

多线程三种实现方式对比

	优点	缺点
继承Thread类	编程比较简单，可以直接使用Thread类中的方法	可以扩展性较差，不能再继承其他的类
实现Runnable接口	扩展性强，实现该接口的同时还可以继承其他的类	编程相对复杂，不能直接使用Thread类中的方法
实现Callable接口		

高级软件人才培训专家

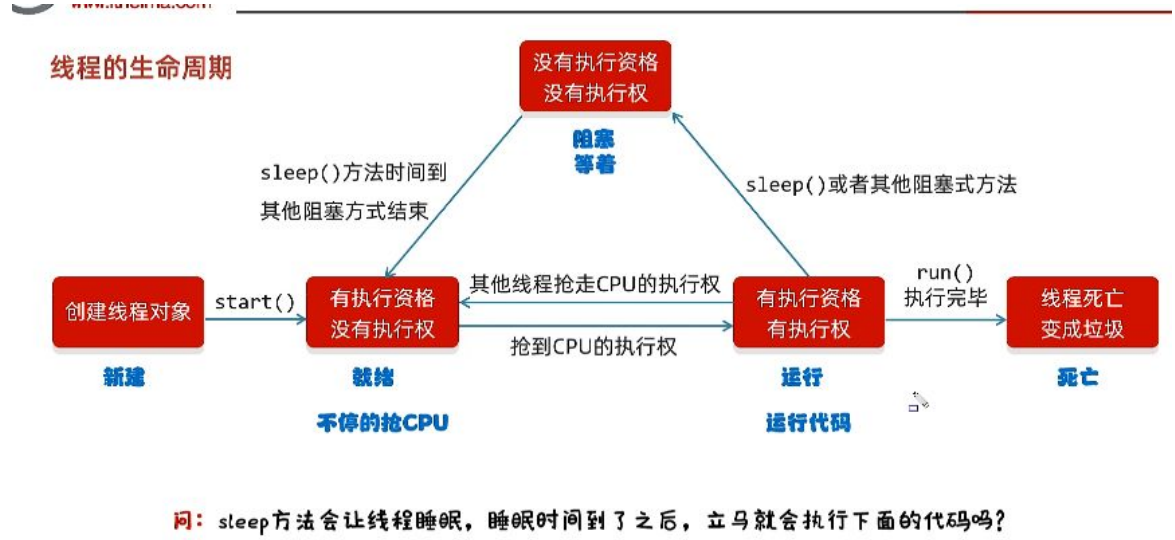
常用成员方法

常见的成员方法

方法名称	说明
String getName()	返回此线程的名称
void setName(String name)	设置线程的名字（构造方法也可以设置名字）
static Thread currentThread()	获取当前线程的对象
static void sleep(long time)	让线程休眠指定的时间，单位为毫秒
setPriority(int newPriority)	设置线程的优先级
final int getPriority()	获取线程的优先级
final void setDaemon(boolean on)	设置为守护线程
public static void yield()	出让线程/礼让线程
public static void join()	插入线程/插队线程

附: 抢占式调度 随机分配(1-10)

线程生命周期



问：sleep方法会让线程睡眠，睡眠时间到了之后，立马就会执行下面的代码吗？

同步代码块

数据安全问题: 线程执行的随机性

synchronized(锁){

　操作共享数据的代码

}

特点1: 锁默认打开,一个线程进去,锁自动关闭

特点2:代码全部执行完毕, 线程出来锁自动打开

同步方法

修饰符 synchronized 返回值类型 方法名(方法参数){

}

锁住方法里的所有代码, 锁对象不能自己指定

Lock锁

虽然我们可以理解同步代码块和同步方法的锁对象问题，
但是我们并没有直接看到在哪里加上了锁，在哪里释放了锁，
为了更清晰的表达如何加锁和释放锁，JDK5以后提供了一个新的锁对象Lock

Lock实现提供比使用synchronized方法和语句可以获得更广泛的锁定操作
Lock中提供了获得锁和释放锁的方法
void lock()：获得锁
void unlock()：释放锁

Lock是接口不能直接实例化，这里采用它的实现类ReentrantLock来实例化
ReentrantLock的构造方法
ReentrantLock()：创建一个ReentrantLock的实例

死锁

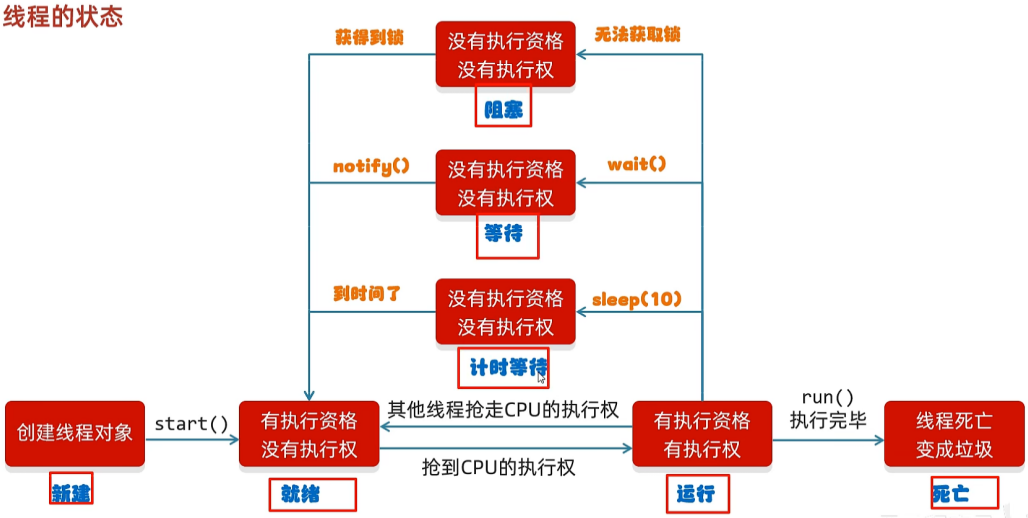
等待唤醒机制

生产者消费者模式是一个经典的多线程协作模式

方法名称	说明
void wait()	当前线程等待，直到被其他线程唤醒
void notify()	随机唤醒单个线程
void notifyAll()	唤醒所有线程

线程的状态

线程的状态



线程的状态

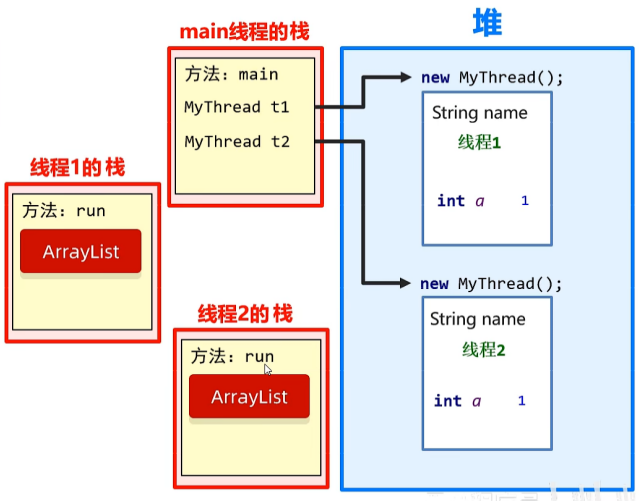
新建状态（NEW）	→	创建线程对象
就绪状态（RUNNABLE）	→	start方法
阻塞状态（BLOCKED）	→	无法获得锁对象
等待状态（WAITING）	→	wait方法
计时等待（TIMED_WAITING）	→	sleep方法
结束状态（TERMINATED）	→	全部代码运行完毕

Java中没有定义运行时的状态

线程栈


```
public static void main(String[] args) {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();
    t1.setName("线程1");
    t2.setName("线程2");
    t1.start();
    t2.start();
}
```

```
public class MyThread extends Thread{
    int a = 1;
    @Override
    public void run() {
        创建ArrayList集合呢?
    }
}
```



线程池

线程池主要核心原理

① 创建一个池子，池子中是空的

② 提交任务时，池子会创建新的线程对象，任务执行完毕，线程归还给池子
下回再次提交任务时，不需要创建新的线程，直接复用已有的线程即可

③ 但是如果提交任务时，池子中没有空闲线程，也无法创建新的线程，任务就会排队等待

Executors

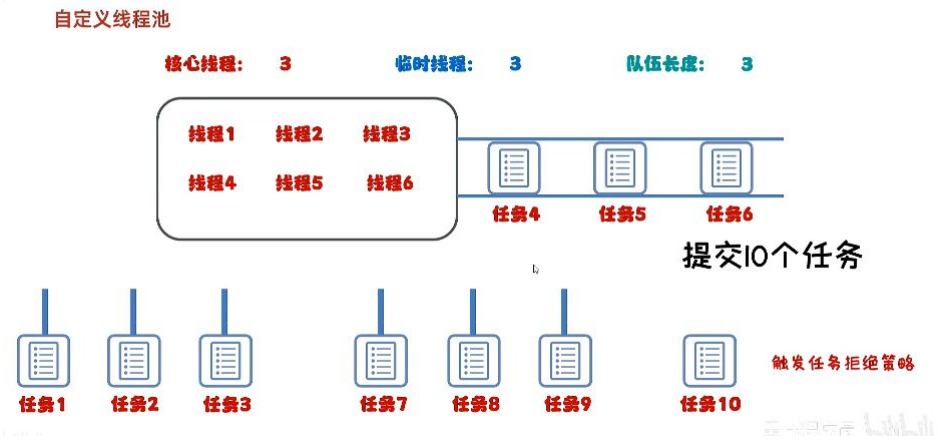
public static ExecutorService newCachedThreadPool() //无上限的线程池

public static ExecutorService newFixedThreadPool(int nThread) //有上限的线程池

相关参数

饭店的故事

核心元素一：正式员工数量 → 核心线程数量
核心元素二：餐厅最大员工数 → 线程池中最大线程的数量
核心元素三：临时员工空闲多长时间被辞退（值） → 空闲时间（值）
核心元素四：临时员工空闲多长时间被辞退（单位） → 空闲时间（单位）
核心元素五：排队的客户 → 阻塞队列
核心元素六：从哪里招人 → 创建线程的方式
核心元素七：当排队人数过多，超出顾客请下次再来（拒绝服务） → 要执行的任务过多时的解决方案



任务拒绝策略

CPU密集型: 主要依赖于处理器性能来完成的任务或程序。计算量大,读取数据操作较少　　视频编码、科学计算、机器学习训练、复杂的图形渲染等都是CPU密集型任务的例子。 最大并行数+1

I/O密集型：取决于I/O系统的性能，如硬盘读写速度、网络带宽等。