多线程实践

一、Java多线程技能

1.1 进程和线程

进程：操作系统的基本执行单元

线程：程序的执行线索

1.2 创建线程的常用方式

1.继承Thread类

2.实现Runnable接口

3.实现Callable接口

1.3 线程安全问题

多线程访问临界资源，破坏原子操作

1.4 停止线程

1. interrupt方法

interrupted:判断当前线程是否中断，并清除中断标志位

isInterrupted:判断线程是否中断，不清除中断标志位

2. 异常抛出法

判断线程处于中断状态后，抛出interruptedException来防止线程继续执行（也可以使用return）

3. stop 方式（弃用）

1.5 暂停线程(弃用)

1.线程挂起suspend（如果一个线程使用打印功能时被挂起，那么其它线程无法使用该打印方法，因为使用了同步机制System.out.print(“hello”)

2.线程恢复 resume

1.6 线程放弃CPU资源

yield()方法

1.7 线程优先级

操作系统中，线程可以划分优先级，优先级高的线程获取cpu时间片的概率也高。线程具有继承性，即线程的优先级与创建它的线程的优先级保持一致。线程又具有随机性，不一定优先级高的线程总是最快的完成任务，但是总体来说优先级高的线程获取cpu资源的概率要大。可以通过setPriority()方法来设置线程的优先级。

1.8 守护线程

Java中存在两种线程，一种是用户线程，一种是守护线程。

守护线程最典型的应用是GC(垃圾回收)thread.setDaemon(true),

当所有的用户线程结束了，守护线程会随着JVM一起结束。

二、对象及变量的并发访问

栈内存独立，堆内存共享。方法中的变量永远是安全的，多线程访问临界资源，破坏了原子操作，就会引发线程安全问题。同步的原则是对共享资源做同步，锁对象必须是同一个对象。

1.synchronized对象监视器为Object

1.当对一个方法加synchronized时，锁对象就是this。当出现异常时，线程会释放掉它拥有的锁对象。同步不具有继承性，即子类无法直接继承父类的同步方法，子类想要同步，必须在子类方法中添加synchronized关键字。

2.Synchronized锁重入：

指的是一个线程在synchronized方法中可以调用该类的另一个synchronized方法，该线程一定可以获取到该锁对象。当存在继承关系时，子类也可以通过可重入锁调用父类上锁（加synchronized）的方法。

3.数据类型String的常量池特性：

同步代码块不使用String作为锁对象。会造成同步，其他线程不执行。

4.多线程死锁问题：

多个线程等待对方的锁释放，造成线程无法继续执行，锁也无法释放，形成死锁。

2.synchronized对象监视器为Class

对象监视器为字节码对象，一定是同一个对象，可以完成同步功能，但是可能会影响效率。Synchronized修饰的方法锁对象是this。Synchronized修饰的static方法锁对象是class字节码对象，两者不同。

3.非线程安全是如何出现的

多线程访问临界资源，破坏了原子操作，造成了数据的不安全性。

4.volatile的主要作用

使变量在多个线程中可见，当线程访问volatile修饰的变量时，强制从公有堆栈中取值。

5.volatile与synchronized的区别

Volatile关键字：只能修饰变量，可以保持变量的可见性，但是不能保证原子性，非阻塞

Synchronized关键字：修饰方法或者代码块，可以保证可见性和原子性，阻塞，synchronized具有volatile的同步功能，它能够将共享内存数据同步到线程的工作内存中。

6.线程安全问题主要研究的两点：

原子性、可见性 （公有堆栈和私有堆栈及其同步问题）

7.使用wait()和notify()方法完成通信

1.wait()使线程等待，notify使等待的线程继续运行。wait()和notify()基于同一个锁对象进行通信。

2.调用wait(）或notify()方法前需要先获取到锁对象，即必须在同步方法或者同步代码块中。获取的锁对象和释放的锁对象必须是同一个即由获取到的锁对象调用wait()或notify()。不然会抛异常。

3.调用wait()方法后，线程释放掉了锁对象进入队列等待状态，等待另一个线程的notify方法。

4.调用notify()方法后，当前线程不会马上释放锁，呈wait状态的线程也不会立马获得锁对象，需要等到当前线程执行完程序即退出synchorined代码块。notify()只随机通知一个线程进行唤醒，notifyAll()唤起所有线程。

5.wait(long) 表示超过这段时间线程没有被唤醒则自动唤醒。

6.过早通知的问题 即当线程先执行notify()方法时可能会引起wait()方法调用后线程一直等待问题。

8.生产者和消费者模式的实现

存在虚假唤醒，比如生产者唤醒生产者，消费者唤醒消费者，使用while（条件）做检查，重复检查条件。

存在所有线程处于wait()状态情况，使用notifyAll()。

1.一生产一消费

2.一生产多消费

3.多生产一消费

4.多生产多消费

9.join()方法的实现

Join()方法表示当调用join()的线程结束后当前线程才继续执行。

Join(long millis)表示等待指定时间后继续执行。

如：在t2线程中调用t1.join()表示暂停t2线程，t1线程执行完后t2线程继续执行。

Join的内部是由wait()实现的，通过判断调用join的线程是否alive或者是否已经过了指定等待的时间。

Join(long) 和 sleep(long)的区别：

Join()内部是wait()实现，会释放锁对象，sleep()不会释放锁对象。注意，这里锁对象指的是线程对象。

10.ThreadLocal类的使用（需要阅读源码）

用于解决每个线程拥有自己的数据。ThreadLocal好比一个Map，key对应着每个线程，value对应着这个线程的数据。

Get()方法：在哪个线程中调用threadLocal.get()方法就是取这个线程对应的值。

Set()方法：设置当前线程的副本变量值。

Remove()方法：移除当前线程的副本变量值。

InitialValue()方法：初始化一个threadlocal的值，每个线程都可以获取到这个值。

注意点：

建议是：每个线程只存一个变量，这样的话所有的线程存放到map中的Key都是相同的ThreadLocal，如果一个线程要保存多个变量，就需要创建多个ThreadLocal，多个ThreadLocal放入Map中时会极大的增加Hash冲突的可能。

1. 线程状态变更图

