多线程学习：

何为死锁？？？

面试题：

<https://www.cnblogs.com/bsjl/p/7693029.html>

<https://www.cnblogs.com/jinlinFighting/p/5657086.html>

<https://blog.csdn.net/ll666634/article/details/78615505>

<https://www.cnblogs.com/dolphin0520/p/3932934.html>

1. 为什么System.out.println效率低下，耗时？？？不能用？同步锁
2. IsAlive 表示线程是否存活，存活表现为正在和准备开始运行。

线程同步状态图：

<https://www.cnblogs.com/mengdd/archive/2013/02/20/2917966.html>

1. 停止线程的方法：

Stop、suspend、resume方法已经被弃用。使用这个会产生不可预料的结果。

1. 使用退出标志，在run方法中break或者return，正常终止线程
2. 使用interrupt方法中断线程，异常方法往上抛

然而interrupt方法并不是真正停止线程，而是在当前线程打一个停止的标志，并不是真正停止。

1. 使用stop暴力释放锁，已经弃用

可能会造成数据不一致

1. Suspend和resume已经弃用

Suspend可能会造成独占锁的情况，即suspend一直没有被resume的情况，导致其他线程不能访问共享数据。

1. Yield方法

Yield方法的作用是放弃当前的cpu时间，将它让给其他的任务去占用CPU执行时间。但放弃的时间不确定，有可能刚刚放弃，马上又获得CPU时间片。

1. 线程的优先级

线程的优先级只是使得优先级高的获得cpu资源的机会高.

1. 守护线程

守护线程是一种特殊的线程，当线程中不存在非守护线程了，则守护线程自动销毁。典典型的守护线程为GC垃圾回收线程。Main线程是用户线程。

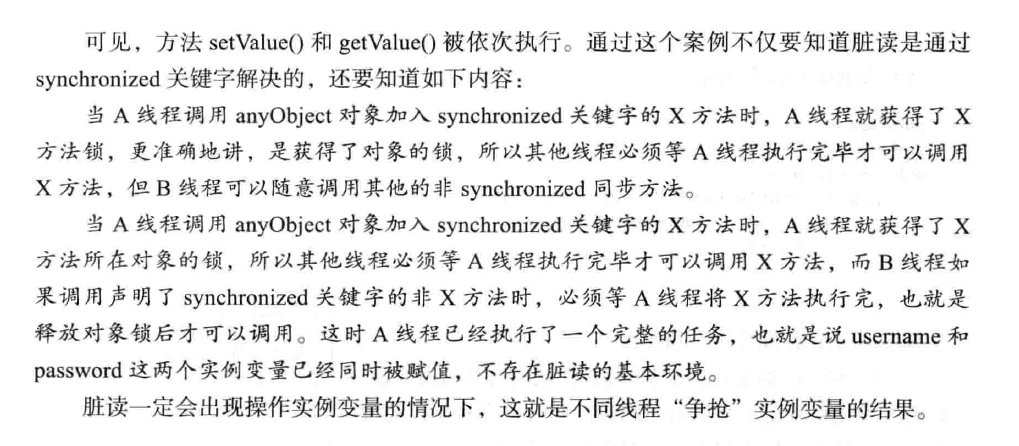
守护线程setDaemon(true)必须在thread.start()之前设置，同时在守护线程中产生新的线程也是守护线程。

1. Synchronized

a)A线程先持有Object对象的lock锁，B线程可以以异步的方式调用Object对象中的非Synchronized类型方法

b)A线程先持有object对象的lock锁，B线程如果在这时调用Object对象中的synchronized类型的方法则需等待，也就是同步。

因此需要格外注意，对于synchronized和synchronized关键字的使用，在同一个对象中，在非synchronized访问中可能会出现脏读的现象。



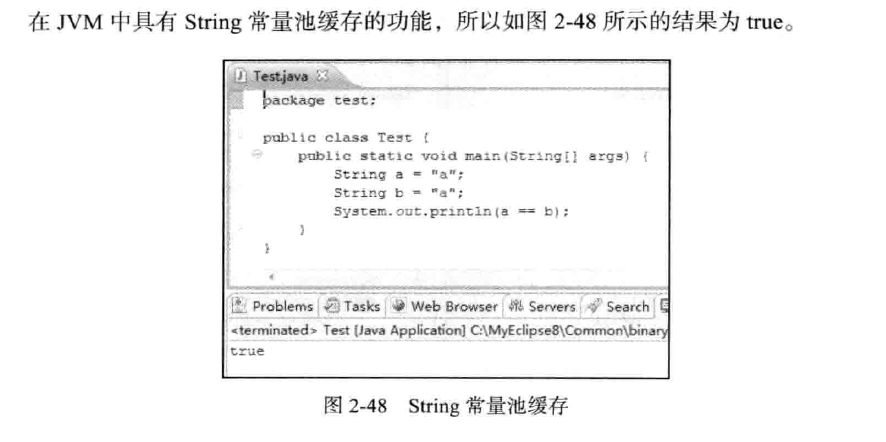
1. Synchronized锁重入

锁重入的大概意思是，在synchronized方法/块中仍然可以调用synchronized修饰的方法，并且是永远可以得到锁的，其实如果不重入的话，就会造成死锁。（外层synchronized方法占用了锁，如果得不到重入，那么synchronized将会一直不释放锁）

另外：锁重入是可以在继承关系中体现的，即子类的synchronized方法调用父类的synchronized方法。

1. 当一个持有锁的线程出现异常时，该线程所持有的锁将会被释放。
2. Synchronized代码块

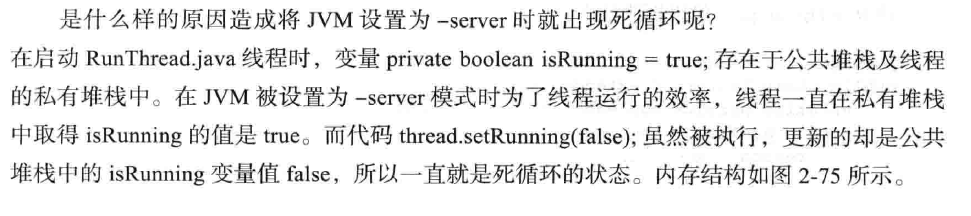
需要注意如下问题

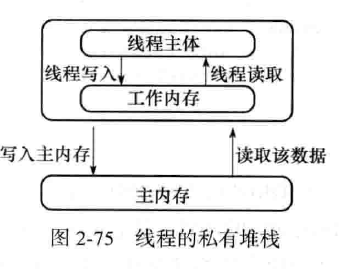


所以，不建议使用string作为锁对象。这样有可能会使得原本预期不是同步的变成同步的过程。

1. 死锁
2. Volatile关键字

Volatile的作用是强制从公共堆栈中取得变量的值，而不是从线程私有数据栈中取得变量的值。





Volatile不能保证原子性。而synchronized可以保证原子性。Volatile关键字使得多个线程共享同一个数据具有可见性，不经常使用。因为synchronized可以达到同样效果，并且具有原子性。

1. 等待/通知机制

Wait方法，必须要同步方法或者同步块中使用，执行该方法后，当前线程释放锁，并且在wait所在的代码处停止执行。如果调用wait时没有持有适当的锁，则抛出异常.

Notify方法，该方法用来通知哪些可能等待该对象的对象锁的其他线程，如果有多个线程等待，则由线程规划器随机挑选一个呈wait状态的线程，对其发出通知notify，并使他等待获取该对象的对象锁。在执行了notify后，当前线程不会马上释放锁，要等待执行完synchronized代码块才会释放。

1. 当interrupt方法遇到wait方法

当线程呈现wait状态时，调用interrupt方法会出现InterruotedException异常

1. Wait(long)

该方法意思是，进入wait后，如果long时间内没有被唤醒，则自动唤醒.

1. 生产者与消费者问题

a)一个生产者和一个消费者：

<https://blog.csdn.net/thinkpadshi/article/details/8163751>

<https://blog.csdn.net/qq_41247433/article/details/79434202> -- 阻塞队列实现

b)多个生产者---一个消费者

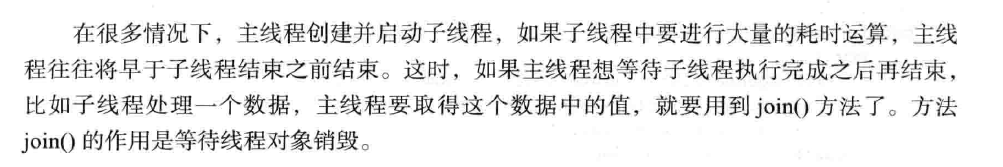
c)多个生产者—多个消费者

d)一个消费者—多个生产者

这些要求检查使用while，和notifyAll

<http://www.cnblogs.com/0201zcr/p/4758533.html>

1. 通过管道进行线程通信？？？？？
2. 使用join方法



1. Join(long) 和 sleep(long)的区别

Join(long)内部调用wait(long)方法，所以释放锁，而sleep不释放锁。

join(0)的意思不是A线程等待B线程0秒，而是A线程等待B线程无限时间直到B线程执行完毕，即join(0)等价于join()。

<https://www.cnblogs.com/lcplcpjava/p/6896904.html>

1. 使用ReentrantLock类

等同于synchronized，且更加灵活

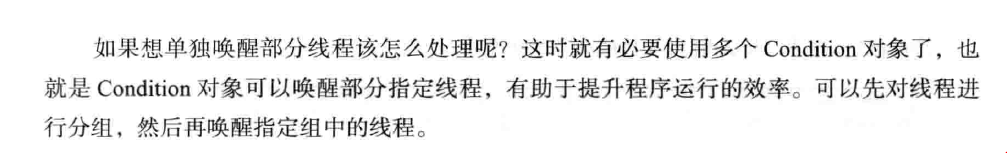
Lock lock = new ReentrantLock();

Lock.lokc();

…

Lock.unlokc();

1. ReentrantLock和Condition实现等待/通知



通熟易懂的说，就是消费者/生产者的场景中，在原来的基础上，增加了队列满时及时通知消费者，队列空时及时通知生产者的优化，通常是两个条件变量一起出现，一个控制值，但两个条件变量可以毫无关系，终归来说还是在Lock的范围内。**所以，从本质上来说，是对Object监视器的场景性优化，而不是全新机制的引入。**

<https://www.cnblogs.com/zhjh256/p/6389168.html>

<https://www.cnblogs.com/sheeva/p/6484224.html> -- condition阻塞队列

Condition实现生产者、消费者？

1. ReentrantReadWriteLock类
2. SimpleDateFormat非线程安全问题
3. 编写工具类，返回一个新的实例解决
4. 使用ThreadLocal

传智播客张孝祥视频学习：

<http://v.baidu.com/v?ct=301989888&rn=20&pn=0&db=0&s=25&ie=utf-8&word=java%20%E5%A4%9A%E7%BA%BF%E7%A8%8B%E8%A7%86%E9%A2%91%20%E7%A5%A5>

或者：<http://www.icoolxue.com/album/show/109>

<https://www.jczhijia.com/CoursePay/93168.html>

1. 传统实现多线程的方式

\* 1.写Thread子类（继承，重写 run）

\* 2.在Thread的构造参数传入 Runnable

第一种： Thread thred = new Thread(){

public void run(){

。。。

}

};

Thred.start();

第二种： thread thread = new Thread(new Runnable(){

public void run(){

。。。

}

})

Thread.start();

这里涉及知识点：

能否在run方法中抛出 InterruptedException e，以便在run中不许要对Thread.sleep(1000);使用try{}catch(){} 捕获？？？

2. JDK定时器的使用

Timer类

TimerTask类

3. 线程的互斥与同步通信

使用synchronized代码块及其原理

互斥指的是同一个对象

这里如果多个线程这部分的Object一样，就会互斥

Synchronized(Object){

}

这样表示同一个对象执行这个代码块时互斥

Synchronized(this){

}

使用synchronized方法

这里相当于synchronized(this)

Public synchronized void A(){

}

//这里相当于字节码级别锁

Public static synchronized void B(){}

面试宝典：子线程循环10次，主线程循环100次。。。

经验：要用到共同数据（包括同步锁）的若干个方法应该归在同一个类身上，这种设计正好体现高类聚和程序的健壮性。锁不是放在线程中的，而是放在资源对象身上。

4 ThreadLocal 线程范围内的共享数据

每个数据都是与当前线程有关的。

一个ThreadLocal代表一个变量，故其中只能放一个数据。如果有两个变量都要线程范围内共享，则要定义两个ThreadLocal或者，封装成一个DTO实体中。每个线程结束的时候，应该调用ThreadLocal.clear()；这样会更快释放内存。

5 jdk5之后的并发库

Java.util.concurrent包及其子包

AtomicInteger操作整数

6 线程池

创建固定大小线程池

创建缓存线程池

创建单一线程池（如何实现线程死掉后重新启动？）

7 用线程池启动定时器

8 jdk并发库线程锁

Lock替代synchronized

读写锁：ReadWriteLock

分为读锁和写锁，多个读锁不互斥，读锁与写锁互斥，写锁与写锁互斥，这是由JVM自己控制的，你只要上好相应的锁即可。如果很多人同时读，但不能同时写，那就上读锁。如果你的代码修改数据，只能有一个人在写，且不能同时读取，那就上写锁。总之，读的时候上读锁，写的时候上写锁！

面试题：设计一个缓存系统？？？

9 Lock&Condition实现线程同步通信

Condition的功能类似在传统线程技术汇总的Object.wait和Object.notify的功能。在等待Condition时，允许发生虚假唤醒，这通常作为对基础平台语义的让步。对于大多数应用程序，这带来的实际影响很小，因为condition应该总是在一个循环中被等待，并测试正被等待的状态声明。某个实现可以随意移除可能的虚假唤醒，但建议应用程序程序员总是假定这些虚假唤醒可能发生，因此总是在一个循环中等待。