

血洗多线程，抱得 offer 归

2019年6月13日 星期四 下午7:43

工作和面试之中，遇到了很多多线程问题。这里我总结了一下，希望对你有所帮助

当然，面试时拿来装逼用，也是极好的。

先来10个。

我来评个级

一、玩命的创建线程池

现象：系统资源耗尽，进程僵死。

原因：每次方法执行，都new一个线程池。

小姐姐味道解决方案：共用一个线程池即可。

作死等级：五颗星

脑残等级：五颗星

```
void doJob(){
```

。本篇内容，基本上都是一些反例，有些很低级但常见。


```
ThreadPoolExecutor exe = new ThreadPoolExecutor(...);
exe.submit(new Runnable(){...})
}
```

复制代码

二、锁泄漏

现象：某个线程一直持有锁而不释放，造成锁泄漏。

原因：未知异常或逻辑导致unlock函数未执行。

小姐姐味道解决方式：始终将unlock函数放在finally中。

作死等级：三颗星

脑残等级：四颗星

```
private final Lock lock = new ReentrantLock();
void doJob(){
    try{
        lock.lock();
        //do. sth
        lock.unlock();
    }catch(Exception e){
    }
}
```

复制代码

三、忘记同步变量

现象：在某个条件下，抛出IllegalMonitorStateException。

原因：调用wait、notify等，忘记synchronized，或者同步了错误的变量。

小姐姐味道解决方式：调用这些函数之前，要使用同步关键字同步它。

作死等级：两颗星

脑残等级：四颗星

```
Object condition = new Object();  
condition.wait();
```

复制代码

四、HashMap死循环

现象：cpu占用高，发生死循环，使用jstack查看是阻塞在get方法上。

原因：在某种条件下，进行rehash时，会形成环形链。某些get请求会走到这个环上

小姐姐味道解决方式：多线程环境下，使用ConcurrentHashMap，别犹豫。

作死等级：四颗星

脑残等级：四颗星

五、给同步的变量重新赋值

现象：不能够达到同步效果，结果是错误的。

原因：非基本类型被重新赋值，会改变锁的指向，不同线程持有的锁可能不一样。

小姐姐味道解决方式：把锁对象声明为final类型。

作死等级：四颗星

脑残等级：三颗星

```
List listeners = new ArrayList();
void add(Listener listener, boolean upsert){
    synchronized(listeners){
        List results = new ArrayList();
        for(Listener ler:listeners){
            ...
        }
        listeners = results;
    }
}
```

复制代码

六、线程循环未捕获异常

现象：线程作业无法继续运行，不明终止。

原因：未捕获循环中的异常，造成线程退出。

小姐姐味道解决方式：习惯性捕获所有异常。

作死等级：三颗星

脑残等级：三颗星

```
volatile boolean run = true;
void loop(){
    while(run){
        //do . sth
        int a = 1/0;
    }
}
```


复制代码

七、volatile误作计数器

现象：多线程计数结果有误。

原因：volatile保证可见性，不保证原子性，多线程操作并不能保证其正确性。

小姐姐味道解决方式：直接使用Atomic类。

作死等级：三颗星

脑残等级：两颗星

```
volatile count = 0;  
void add(){  
    ++count;  
}
```

复制代码

八、错误保护范围

现象：虽然使用了线程安全的集合，但达不到同步效果。

原因：操作要修改多个线程安全的集合，但操作本身不是原子的。

小姐姐味道解决方式：弄明白要保护的代码逻辑域。

作死等级：三颗星

脑残等级：四颗星


```
private final ConcurrentHashMap<String,Integer> nameToNumber;  
private final ConcurrentHashMap<Integer,Salary> numberToSalary;  
public int geBonusFor(String name) {  
    Integer serialNum = nameToNumber.get(name);  
    Salary salary = numberToSalary.get(serialNum);  
    return salary.getBonus();  
}
```

复制代码

再比如下面的错误代码。

```
Map<String, String> map = Collections.synchronizedMap(new HashMap<S  
if(!map.containsKey("foo"))  
    map.put("foo", "bar");
```

复制代码

九、一些老的日期处理类

现象：使用全局的Calendar,SimpleDateFormat等进行日期处理，发生异常或者数据不

原因：这俩东西不是线程安全的，并发调用会有问题。

小姐姐味道解决方案：放在ThreadLocal中，建议使用线程安全的DateTimeFormatter

作死等级：三颗星

脑残等级：三颗星

```
SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss"  
Date dododo(String str){  
    return format(str);  
}
```

复制代码


```
tring, String>());
```

不准确。

。

```
");
```


十、代码死锁

现象：代码产生死锁和相互等待。

原因：代码满足了下面四个条件：互斥；不可剥夺；请求和保持；循环等待。

小姐姐味道解决方式：破坏这四个条件。或者少用同步。

作死等级：两颗星

脑残等级：一颗星

下面是一段简单的死锁代码。

```
final Object lock1 = new Object();
final Object lock2 = new Object();
new Thread(new Runnable() {
    @Override
    public void run() {
        sleep(1000);
        synchronized (lock1) {
            synchronized (lock2) {
            }
        }
    }
}).start();
new Thread(new Runnable() {
    @Override
    public void run() {
        synchronized (lock2) {
            sleep(1000);
            synchronized (lock1) {
            }
        }
    }
}).start();
```

复制代码

十一、long变量读取无效值

现象：会读取到非设置的值。

原因：long变量读写不是原子的，可能会读到1个变量的高32位和另一个变量的低32位。

小姐姐味道解决方案：确保long和double变量的数据正确，可以加上volatile关键字。

作死等级：一颗星

脑残等级：没有星

扩展阅读(jdk10)：[docs.oracle.com/javase/spec...](https://docs.oracle.com/javase/10/specs/jvms/)

咦？怎么有11个？一定是多线程计算错误。

End

许多java开发，都是刚刚接触多线程开发。但即使是有经验的开发，也会陷入很多误区。

多线程的使用是及其复杂的，使用低级api出错的概率会成倍增加，对技能要求也较高。小姐姐味道这里一个比较浅显但全面的总结：[JAVA多线程使用场景和注意事项简版](#)，但健

2位字节。

多线程的陷阱。当你的程序没有得相应的期望，希望本文能帮你了解到其中的微妙之处。

交高。所幸，concurrent包使得这个过程方便了很多，但依然存在资源规划和同步失效的问题。壮壮的代码还要靠你自己去实践呀。

。小姐姐