1.临界资源问题:在同一个进程中有多个线程执行同一任务,有一个共享资源,当一个线程操作 共享资源时,还没来得及修改,另一个线程把cpu时间片抢去了,又来操作共享资源, 这时就出现.临界资源问题.

2.解决临界资源问题,要用到线程同步.

3.线程同步:让想一起执行代码绑定成一个代码块,一个线程进去执行这个代码块,其他线程就 不能进去,只能在外面等待,等待代码块中线程执行完了,让代码块共享资源让出来, 哪个 线程抢到cpu时间片就可以进入代码块中执行了,这就是线程同步.

4.线程同步的方式:

4.1:同步代码块:有一个线程进入同步代码块自动上锁,这个线程对象把同步代码块中代 码执行完了就会自动解锁.

4.1.1:语法:synchronized (锁对象) {

想要一起执行代码块;

}

4.1.2:同步代码块锁对象可以是任何对象,只要这个锁对象是这多个线程共享,锁对 象的值一般不变.

4.1.3:同步代码块中锁范围越小越好.

eg:public class MyRunnable implements Runnable {

```
//声明一个成员变量存票数
public int ticket=1000;

//创建一个锁对象,所有线程共享
Object ob=new Object();

/**
 * 实现父类的任务方法
 */
@Override
public void run() {
    while (true) {//ticket=1
```

```
                //1,2,3
                //同步代码块
                synchronized (ob) {//1,2
                    if (ticket>0) {

System.out.println(Thread.currentThread().getName()+"正在销售
第"+ticket+"张票");
                        ticket--;
                    }else {

System.out.println(Thread.currentThread().getName()+"票已经销售完
毕");
                        break;
                    }
                }
            }
        }
}
```

4.2:同步方法:将要一起执行代码放在同一个同步方法中,当一个线程进入同步方法会自 动上锁,当这个线程将同步方法中代码全部执行完了,就会自动解锁,此时 其他线程抢到cpu时间片进入同步方法.

4.2.1:同步方法:public synchronized void 方法名(形参列表){
方法体;
}

4.2.2:锁的范围越小越好.

eg:public class MyRunnable implements Runnable {

```
    //声明一个成员变量存票数
    public int ticket=1000;

    /**
     * 实现父类的任务方法
     */
    @Override
```

```
        public void run() {
            while (true) {//ticket=1
                if (saleTicket()) {
                    break;
                }
            }
        }

        /**
         * 同步方法
         */
        private synchronized boolean saleTicket() {
            if (ticket>0) {
                System.out.println(Thread.currentThread().getName()+"正在销
售第"+ticket+"张票");
                ticket--;
                return false;
            }else {
                System.out.println(Thread.currentThread().getName()+"票已经
销售完毕");
                return true;
            }
        }
}
```

4.3:对象互斥锁:要手动上锁,还要确保在任意情况下能解锁.

4.3.1:语法://创建对象互斥锁的对象

```
Lock l=new ReentrantLock();
    //上锁
 l.lock();
//解锁
l.unlock();
```

4.3.2:锁范围越小越好.

4.3.3:对象互斥锁因为是手动上锁,所以一定要确保在任何情况下能解锁,否

则会出现 死锁现象.

5.线程同步:保证数据的安全性.

线程的安全性越高,效率越低.线程的安全性越低,效率越高.

优点:保证数据的安全性.

缺点:效率低(线程不光要捡到CPU时间片,还要抢到锁才能进行锁范围去执行,所以等待的 时间更久,效率低).