1. 互斥锁，自旋锁，原子操作，CAS

**互斥锁：**一个线程在执行A代码块的时候，另一个线程无法进入，旦另一个线程可以切换去做其他事情

**自旋锁：**同上，但另一个线程会自旋等待，不会线程切换做其他事情

当A代码块耗时少，自旋锁好，耗时多，互斥锁好

**原子操作：**一段指令能完成一系列功能，不可在分割。需要指令集支持

**CAS：**底层提供的一段原子操作 CAS(A,B,C) 如果A等于B，则把C赋值给A。 这段判断是原子性的。 （类似于乐观锁的应用）

1. 多线程单例模式实现

private static SingletonDemo instance = null;

//DCL (Double Check Lock 双端捡锁机制）

public static SingletonDemo getInstance() {

if (instance == null) {

synchronized (SingletonDemo.class) {

if (instance == null) {

instance = new SingletonDemo();

}

}

}

return instance;

}

问题：JVM会进行指令重排，可能会锁不住（指令重排只会保证串行语义执行的一致性（单线程），但并不会关心多线程间的语义一致性）

解决方案：使用volatile修饰instance，禁止指令重排

instance=newSingDemo()；可以分为以下3步完成（伪代码）

memory=allocate(): // 1.分配对象内存空间

instance(memory): // 2.初始化对象

instance=memory; //3. 设置instance指向刚分配的内存地址，此时instance != null

步骤2和步骤3不存在数据依赖关系，而且无论重排前还是重排后程序的执行结果在单线程中并没有改变，因此这种重排优化是允许的。

memory=allocate();//1．分配对象内存空间

instance=memory;//3．设置ins怡nce指向刚分配的内存地址，此时instance != null,但是对象还没有初始化完成！

instance(memory);//2．初始化对象