管程　　Monitor 监视器　　所说锁

是一种同步机制，保证同一个时间，只有一个线程访问被保护数据或者代码

jvm同步基于进入和退出，使用管程对象实现的

用户线程：自定义线程　　主线程结束了，用户线程还在运行，jvm存活

守护线程：比如垃圾回收　　没有用户线程了，都是守护线程，jvm结束

创建线程多种方式

第一种 继承Thread类

第二种 实现Runnable接口

第三种 Callable接口

第四种 线程池方式

Runnable接口 和 Callable接口

（1）是否有返回值
（2）是否抛出异常
（3）实现方法名称不同，一个是run方法，一个是call方法

多线程编程步骤

上部

中部

下部

第一步 创建资源类，在资源类创建属性和操作方法

第二步 在资源类操作方法
（1）判断
（2）干活
（3）通知

第三步 创建多个线程，调用资源类的操作方法

第四步 防止虚假唤醒问题

```
add() {
  变量是0 才+1

}
```

例子：
有两个线程

实现对一个初始值是0的变量

一个线程对值 +1

另外一个线程对值 -1

a 1　b 0
a 1　b 0
a 1　b 0

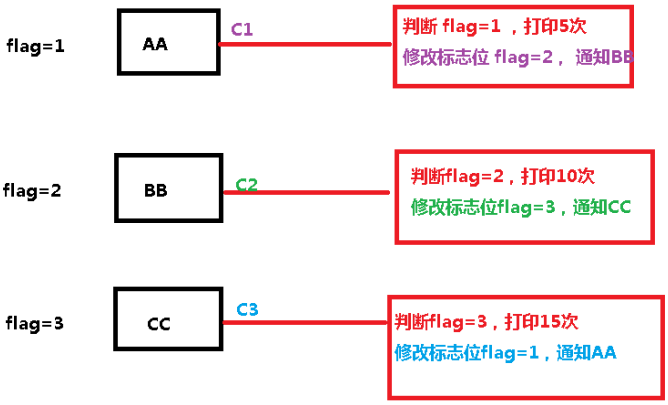
线程间定制化通信

启动三个线程，按照如下要求：

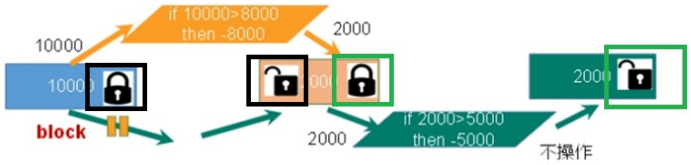
* AA打印5次，BB打印10次，CC打印15次

* AA打印5次，BB打印10次，CC打印15次

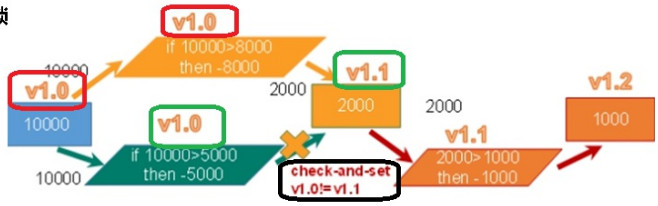
进行10轮

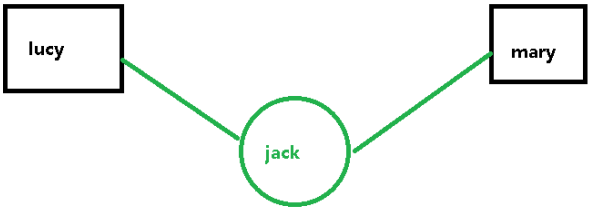


悲观锁



乐观锁





找一个类，既和Runnable有关系，又和Callable也有关系

* Runnable接口有实现类 FutureTask
* FutureTask构造可以传递Callable

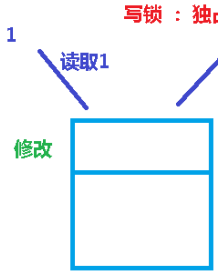
线程1往阻塞队列里添加元素，线程2从阻塞队列里移除元素

表锁

行锁

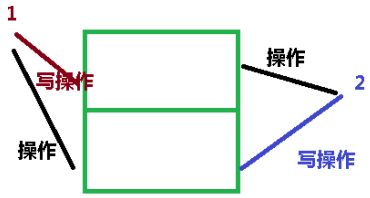
读锁 ： 共享锁，发生死锁

写锁 ： 独占锁，发生死锁

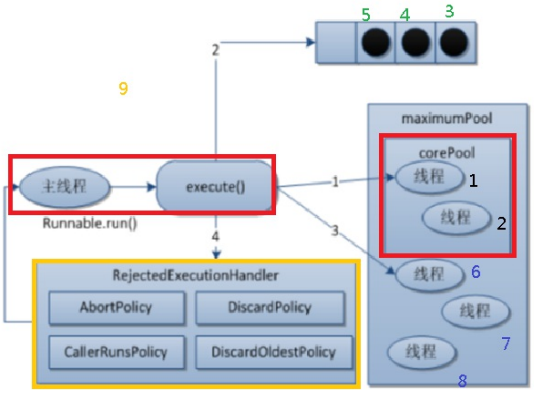


1线程修改时候，需要等待2线程读之后

2线程修改时候，需要等1线程读之后




```
int corePoolSize,                常驻线程数量（核心）
int maximumPoolSize,             最大线程数量
long keepAliveTime,              线程存活时间
TimeUnit unit,
BlockingQueue<Runnable> workQueue,   阻塞队列
ThreadFactory threadFactory,         线程工厂
RejectedExecutionHandler handler     拒绝策略
```

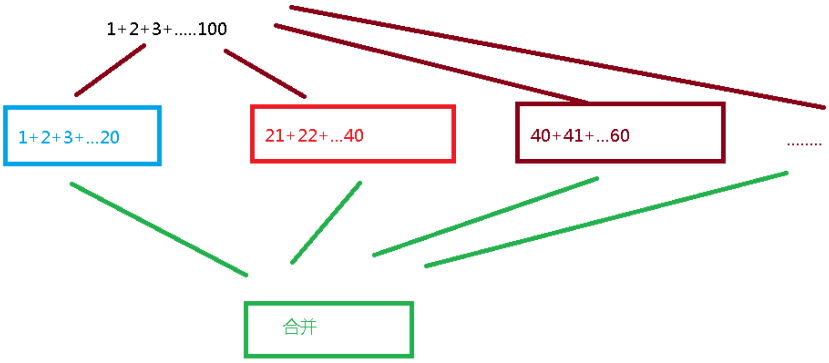


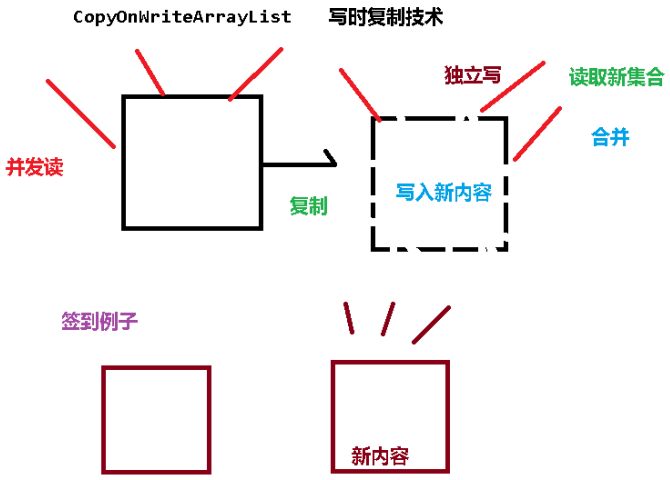
阻塞队列 3

常驻线程数 2

最大线程数 5

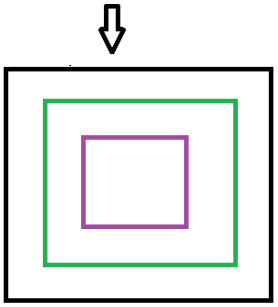
拒绝策略





可重入锁　　synchronized（隐式）和 Lock（显式）都是可重入锁

递归锁

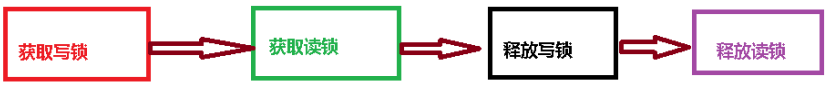


读写锁：一个资源可以被多个读线程访问，或者可以被一个写线程访问，但是不能同时存在读写线程，读写互斥，读读共享的

第一 无锁	第二 添加锁 使用synchronized和ReentrantLock	第三 读写锁 ReentrantReadWriteLock
多线程抢夺资源	都是独占的， 每次只能来一个操作	读读 可以共享，提升性能， 同时多人进行读操作
乱	读读 1，不能共享	写写 1
	读写 1	缺点： （1）造成锁饥饿，一直读，没有写操作，比如坐地铁 （2）读时候，不能写，只有读完成之后，才可以写，写操作可以读
	写写 1	

锁降级：　　将写入锁降级为读锁

jdk8说明：　　写锁　降级　读锁

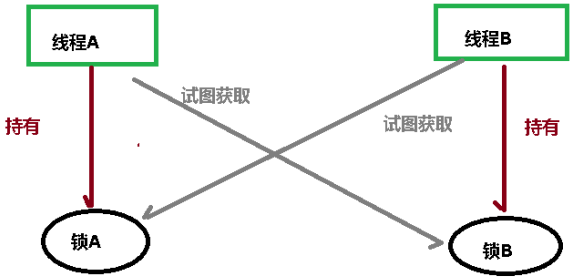


读锁 不能升级为 写锁

增加修改删除--- 写

查询　　--- 读

1、什么是死锁： 两个或者两个以上进程在执行过程中，因为争夺资源而造成一种互相等待的现象，如果没有外力干涉，他们无法再执行下去



2、产生死锁原因： 第一 系统资源不足
第二 进程运行推进顺序不合适
第三 资源分配不当

3、验证是否是死锁

（1）jps 类似linux ps -ef （2）jstack jvm自带堆栈跟踪工具