**死锁：**

**概念**

在多道程序环境中，多个进程可以竞争（不是抢占）有限数量的资源。当一个进程申请资源时，如果这时没有可用资源，那么这个进程进入等待状态。有时，如果所申请的资源被其他等待进程占有，那么该等待进程有可能再也无法改变状态。这种情况称为**死锁**。

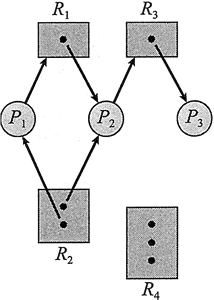
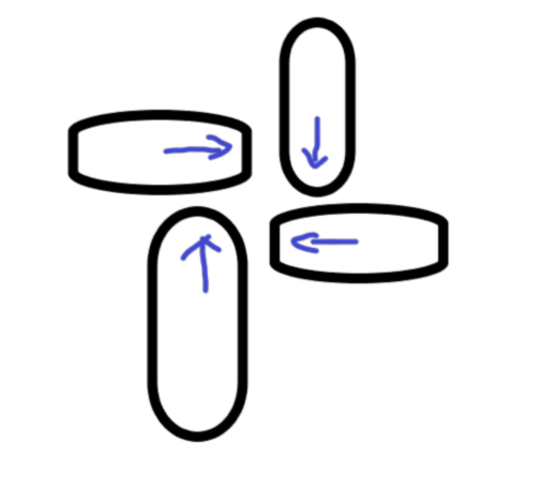
****

图2 抽象图

图1 资源分配图

图1中，形成环P1—> R1 —> P2 一> R3 —> P3 —> R2 —> P1形成死锁。

在正常操作模式下，进程只能按如下顺序使用资源：

* 申请：进程请求资源。如果申请不能立即被允许（例如，申请的资源正在被其他进程使用)，那么申请进程应等待，直到它能获得该资源为止。
* 使用：进程对资源进行操作（例如，如果资源是打印机，那么进程就可以在打印机上打印了）。
* 释放：进程释放资源。

## 死锁特征

#### 必要条件

如果在一个系统中以下四个条件同时成立，那么就能引起死锁：

1. 互斥：至少有一个资源必须处于非共享模式，即一次只有一个进程可使用。如果另一进程申请该资源，那么申请进程应等到该资源释放为止。
2. 占有并等待：—个进程应占有至少一个资源，并等待另一个资源，而该资源为其他进程所占有。
3. 非抢占：资源不能被抢占，即资源只能被进程在完成任务后自愿释放。
4. 循环等待：有一组等待进程 {P0，P1，…，Pn}，P0 等待的资源为 P1 占有，P1 等待的资源为 P2 占有，……，Pn-1 等待的资源为 Pn 占有，Pn 等待的资源为 P0 占有。

我们强调所有四个条件必须同时成立才会出现死锁。循环等待条件意味着占有并等待条件，这样四个条件并不完全独立。

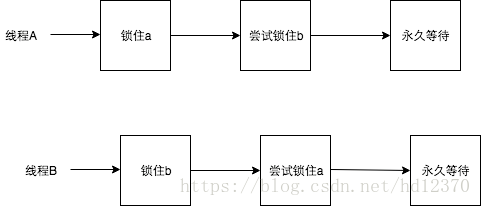
## 解决死锁的基本方法

#### 预防死锁：

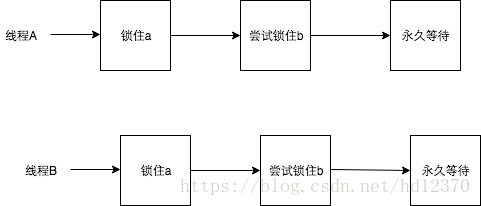
* 资源一次性分配：一次性分配所有资源，这样就不会再有请求了：（破坏请求条件）
* 只要有一个资源得不到分配，也不给这个进程分配其他的资源：（破坏请保持条件）
* 可剥夺资源：即当某进程获得了部分资源，但得不到其它资源，则释放已占有的资源（破坏不可剥夺条件）
* 资源有序分配法：系统给每类资源赋予一个编号，每一个进程按编号递增的顺序请求资源，释放则相反（破坏环路等待条件）

1、以确定的顺序获得锁

如果必须获取多个锁，那么在设计的时候需要充分考虑不同线程之前获得锁的顺序。按照上面的例子，两个线程获得锁的时序图如下：



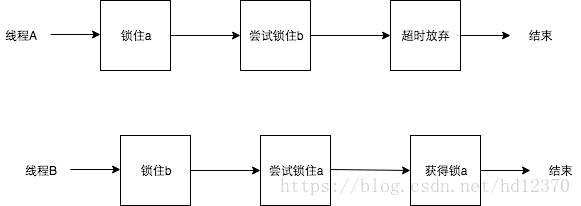
 如果此时把获得锁的时序改成：



那么死锁就永远不会发生。 针对两个特定的锁，开发者可以尝试按照锁对象的hashCode值大小的顺序，分别获得两个锁，这样锁总是会以特定的顺序获得锁，那么死锁也不会发生。问题变得更加复杂一些，如果此时有多个线程，都在竞争不同的锁，简单按照锁对象的hashCode进行排序（单纯按照hashCode顺序排序会出现“环路等待”），可能就无法满足要求了，这个时候开发者可以使用银行家算法，所有的锁都按照特定的顺序获取，同样可以防止死锁的发生，该算法在这里就不再赘述了，有兴趣的可以自行了解一下。

2、超时放弃

当使用synchronized关键词提供的内置锁时，只要线程没有获得锁，那么就会永远等待下去，然而Lock接口提供了boolean tryLock(long time, TimeUnit unit) throws InterruptedException方法，该方法可以按照固定时长等待锁，因此线程可以在获取锁超时以后，主动释放之前已经获得的所有的锁。通过这种方式，也可以很有效地避免死锁。 还是按照之前的例子，时序图如下：



## 避免死锁:

* 预防死锁的几种策略，会严重地损害系统性能。因此在避免死锁时，要施加较弱的限制，从而获得 较满意的系统性能。由于在避免死锁的策略中，允许进程动态地申请资源。因而，系统在进行资源分配之前预先计算资源分配的安全性。若此次分配不会导致系统进入不安全的状态，则将资源分配给进程；否则，进程等待。其中最具有代表性的避免死锁算法是银行家算法。
* 银行家算法：首先需要定义状态和安全状态的概念。系统的状态是当前给进程分配的资源情况。因此，状态包含两个向量Resource（系统中每种资源的总量）和Available（未分配给进程的每种资源的总量）及两个矩阵Claim（表示进程对资源的需求）和Allocation（表示当前分配给进程的资源）。安全状态是指至少有一个资源分配序列不会导致死锁。当进程请求一组资源时，假设同意该请求，从而改变了系统的状态，然后确定其结果是否还处于安全状态。如果是，同意这个请求；如果不是，阻塞该进程知道同意该请求后系统状态仍然是安全的。

## 检测死锁

1. 首先为每个进程和每个资源指定一个唯一的号码；
2. 然后建立资源分配表和进程等待表。

## 解除死锁:

当发现有进程死锁后，便应立即把它从死锁状态中解脱出来，常采用的方法有：

* 剥夺资源：从其它进程剥夺足够数量的资源给死锁进程，以解除死锁状态；
* 撤消进程：可以直接撤消死锁进程或撤消代价最小的进程，直至有足够的资源可用，死锁状态.消除为止；所谓代价是指优先级、运行代价、进程的重要性和价值等。

**银行家算法：**

**算法思想**

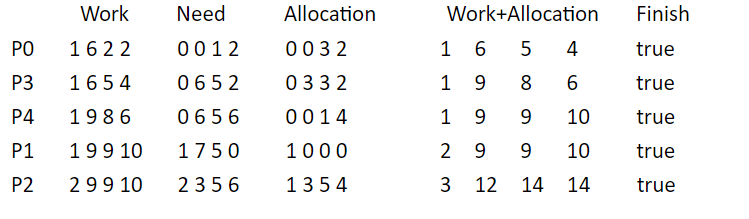
银行家算法：银行家算法是从当前状态出发，按照系统各类资源剩余量逐个检查各进程需要申请的资源量，找到一个各类资源申请量均小于等于系统剩余资源量的进程P1。然后分配给该P1进程所请求的资源，假定P1完成工作后归还其占有的所有资源，更新系统剩余资源状态并且移除进程列表中的P1，进而检查下一个能完成工作的客户，......。如果所有客户都能完成工作，则找到一个安全序列，银行家才是安全的。若找不到这样的安全序列，则当前状态不安全。

**相关数据结构**

1. **可利用资源向量Available**。这是一个含有m个元素的数组，其中的而每一个元素代表一类可利用资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态的改变。如果Available[j]=K,则表示系统中现有Rj类资源K个。
2. **最大需求矩阵Max**。这是一个n\*m的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max[i,j]=K；则表示进程i需要Rj类资源的最大数目为K。
3. **分配矩阵Allocation**。这也是一个n\*m的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果Allocation[i,j]=K，则表示进程i当前已分得Rj类资源的数目为K。
4. **需求矩阵Need**。这也是一个n\*m的矩阵，用以表示每一个进程尚需的各类资源数。如果Need[i,j]=K,则表示进程i还需要Rj类资源K个，方能完成任务。

上述三个矩阵间存在下述关系：Need[i,j]=Max[i,j]-Allocation[i,j]

最后得出的分配顺序：（work是剩余资源，即Available）



全为true，则表示没有死锁，可避免死锁。