Java多线程中的同步与死锁

第 9 组

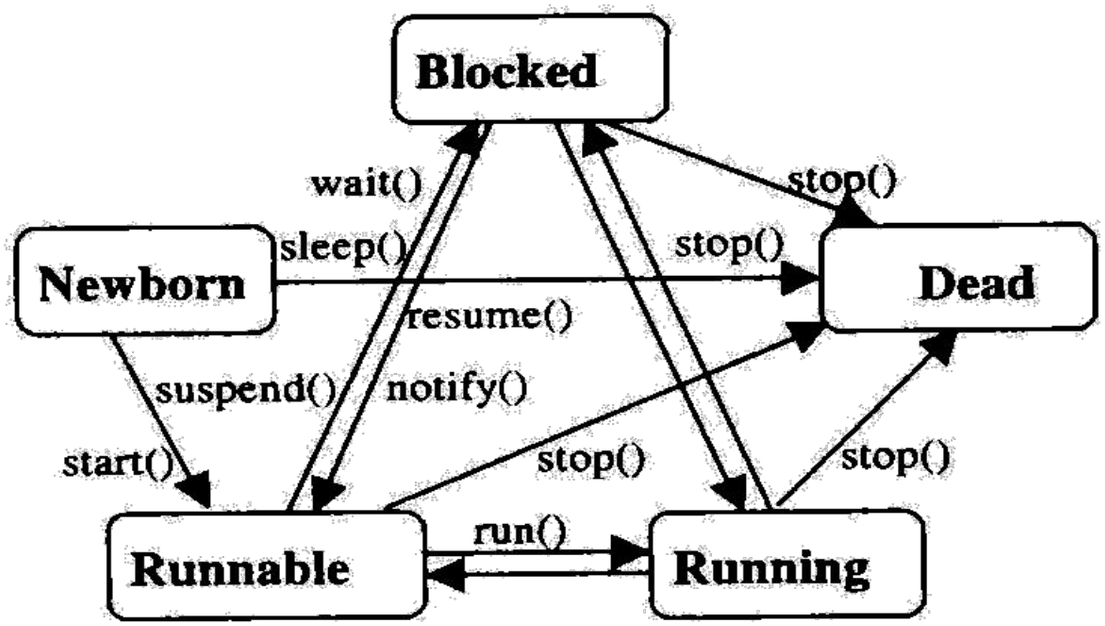
主讲：黄继升

演示：陈乐聪

2019.6.3

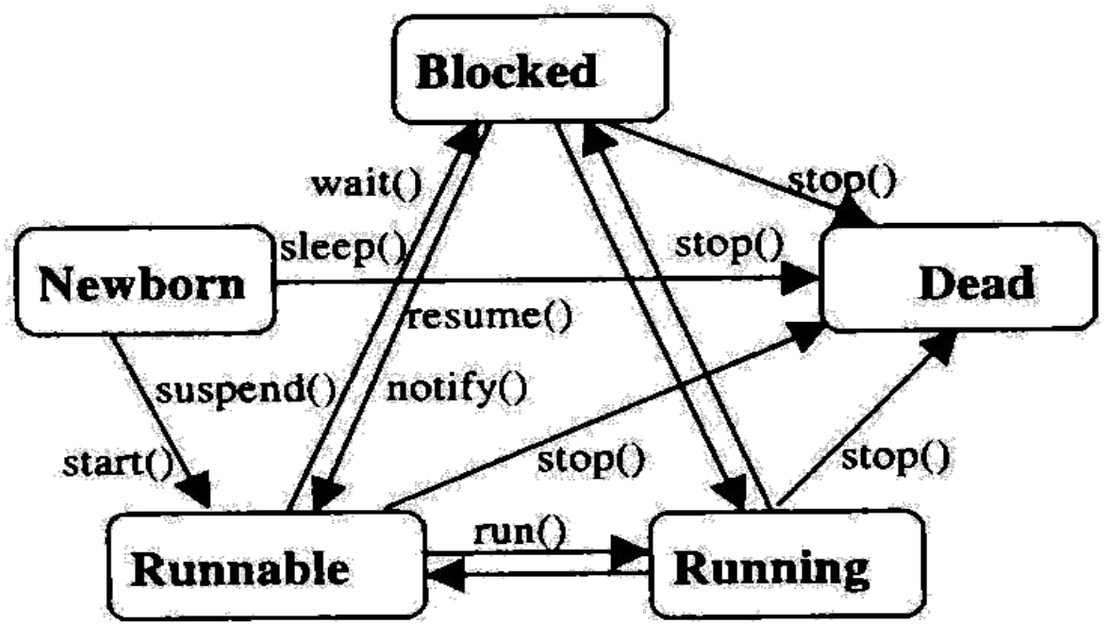
Java线程的生命周期

- 新生态（Newborn）



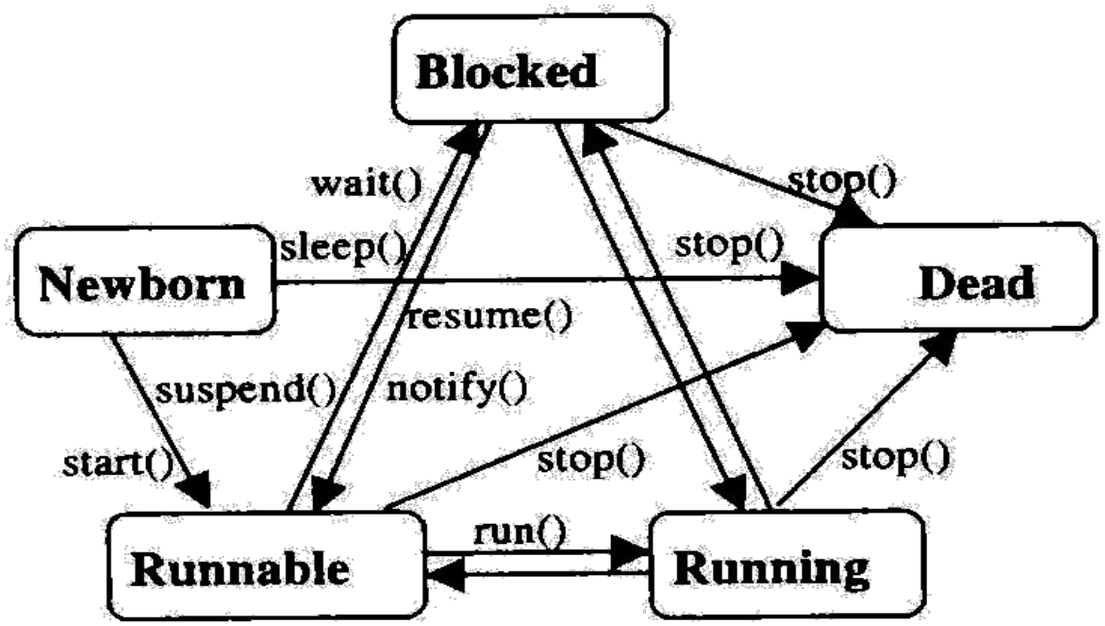
Java线程的生命周期

- 新生态（Newborn）
- 就绪态（Runnable）



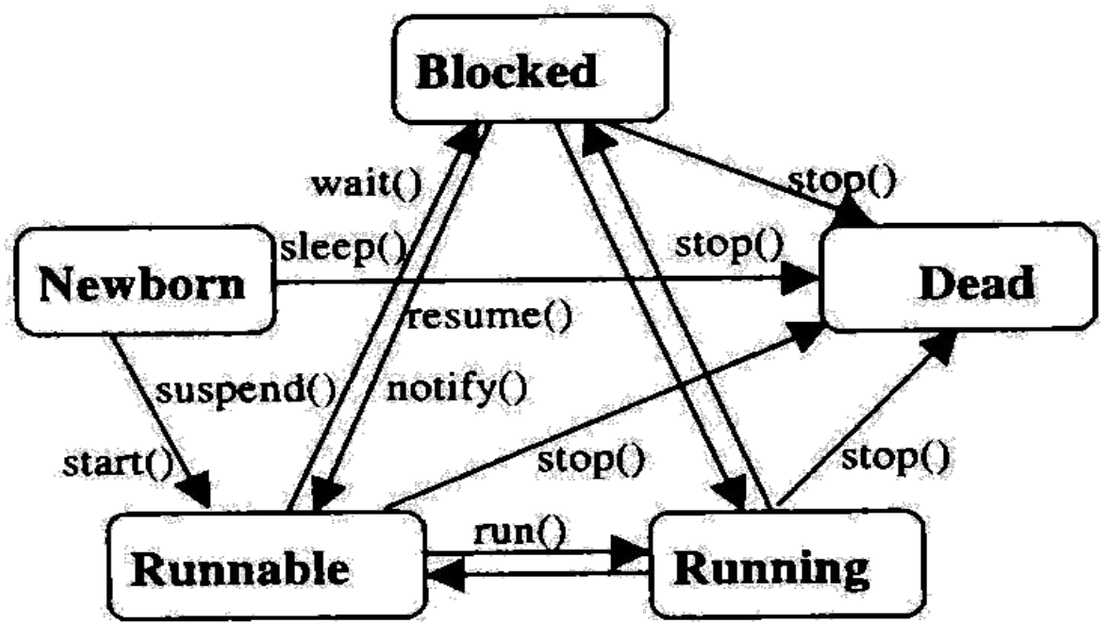
Java线程的生命周期

- 新生态（Newborn）
- 就绪态（Runnable）
- 运行态（Running）



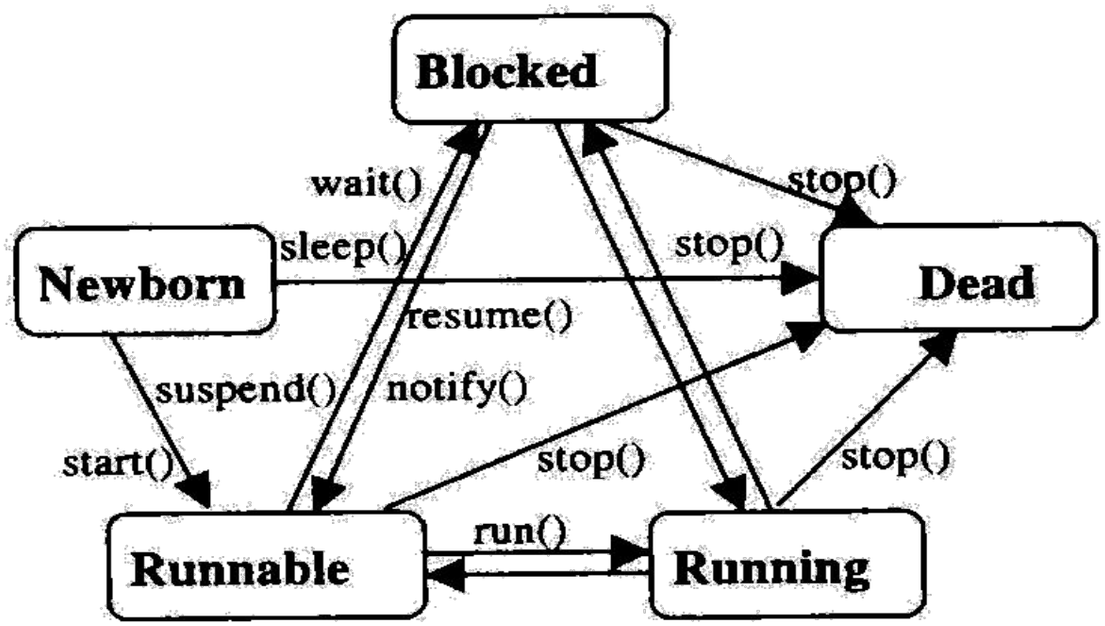
Java线程的生命周期

- 新生态（Newborn）
- 就绪态（Runnable）
- 运行态（Running）
- 阻塞态（Blocked）



Java线程的生命周期

- 新生态（Newborn）
- 就绪态（Runnable）
- 运行态（Running）
- 阻塞态（Blocked）
- 死亡态（Dead）



Java线程的创建方式

- 继承Thread子类

```
public MyThread extends T hread
{
  …//相关属性和方法定义
  public void run()
  {
    … // 线程体代码
  }
public static void main(String args[])
{
   Thread t =new MyThread();
   …
   t.start();//启动线程 t
```

Java线程的创建方式

- 继承Thread子类
- 实现Runnable接口

```
public class MyThread implements Runnable
{
  …//相关属性定义
  public void run()
  {
   … // 线程所要执行操作的代码
  }
}
```

启动的方法有两种，但其实质还是一致的，如下所示：

```
MyThread my = new MyThread();
Thread t = new Thread(my);
t.start();
```

或

```
MyThread my = new MyThread();
new Thread(my).start ;
```

Java线程的创建方式

- 继承Thread子类

  层次清晰，逻辑分明，使用简便

Java线程的创建方式

- 继承Thread子类

  层次清晰，逻辑分明，使用简便

- 实现Runnable接口

  可以弥补第一种方法的不足，即可以实现多重继承

  (Java 中的类只能单重继承)

Java并发机制的分析与研究

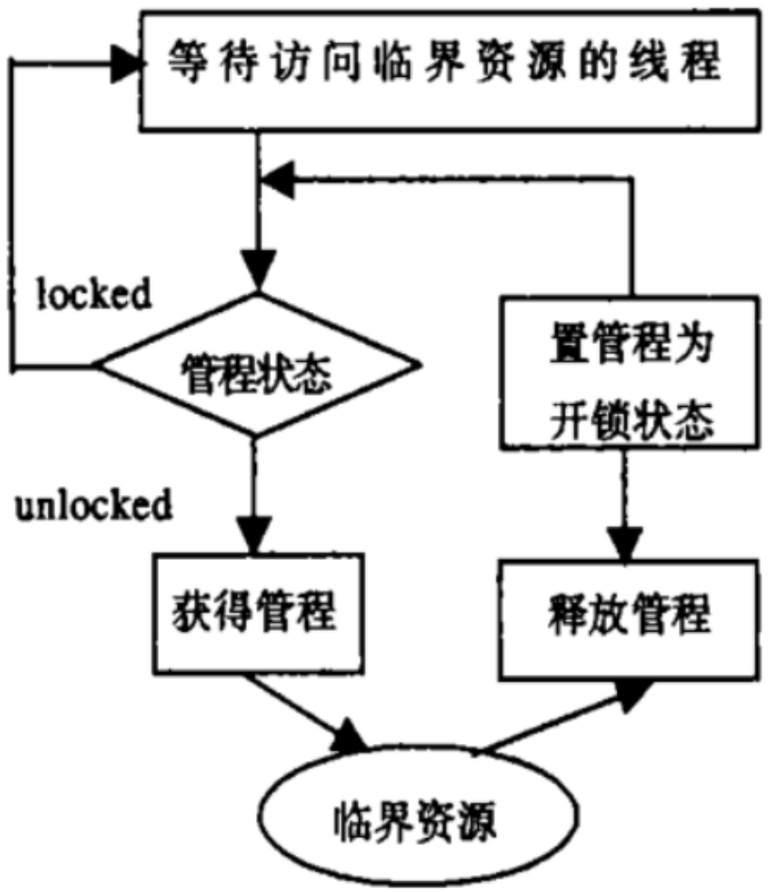
- 基于synchronized和monitor的互斥机制

Java并发机制的分析与研究

- 基于synchronized和monitor的互斥机制
- volatile 关键字解析

Java并发机制的分析与研究

- 基于synchronized和monitor的互斥机制
- volatile 关键字解析
- 基于wait()和notify()的交互通信机制



Java并发机制的分析与研究

- 基于synchronized和monitor的互斥机制
- volatile 关键字解析
- 基于wait()和notify()的交互通信机制
- 通过互斥和交互通信实现同步

```
synchronized(this)                    …
{   …                               P(empty)
      wait();                         P(mutex)
      …                                  CS;
      notify();                       V(mutex)
      …                                P(full)
}                                       …
```

图 3 Java 同步机制与操作系统同步机制的比较图

Java死锁的分析与研究

- 获取锁🔒的持有者算法

1. 当进程加锁的时候, 将<pid, lock_type, resource_ addr>插入死锁检测程序维护的链表 HOLDER_LIST 中。
2. 当进程解锁时, 将<pid, lock_type, resource_ addr>从 HOLDER_LIST 链表中删除. 那么 HOLDER_ LIST 链表中保存的便是那些只有成功加锁而没有解锁的进程。
3. 当死锁检查周期 search_cycle 时间到的时候,便可以从 HOLDER_LIST 中获得锁的持有者。

Java死锁的分析与研究

- 获取锁🔒的持有者算法

```
1 when thread lock
2        insert <pid, lock_type, resource_addr> to HOLDER_LIST
3 when thread unlock
4        delete <pid, lock_type, resource_addr> from HOLDER_LIST
5 while (1){
6        sleep(search_cycle)
7        if HOLDER_LIST not NULL
8              search the lock holders from HOLDER    _LIST
9 }
```

图5 获得锁持有者算法

Java死锁的分析与研究

- 获取锁🔒的等待者算法

1. 异常进程：等待时间超过阈值threshold_time的进程。

Java死锁的分析与研究

- 获取锁🔒的等待者算法

1. 异常进程：等待时间超过阈值threshold_time的进程。
2. 计算等待时间：使用进程描述符 task_struct 结构体中的last_arrival 为参考，来判断进程的等待(包括忙等)时间是否达到 threshold_time。

Java死锁的分析与研究

- 获取锁🔒的等待者算法

1. 异常进程：等待时间超过阈值threshold_time的进程。
2. 计算等待时间：使用进程描述符 task_struct 结构体中的last_arrival 为参考，来判断进程的等待(包括忙等)时间是否达到 threshold_time。
3. 计算阈值threshold_time：threshold_time 必须大于进程持有锁的时间 lock_time，否则会将正常持有锁的进程判为异常进程。

Java死锁的分析与研究

- 获取锁🔒的等待者算法

1. 异常进程：等待时间超过阈值threshold_time的进程。
2. 计算等待时间：使用进程描述符 task_struct 结构体中的last_arrival 为参考，来判断进程的等待(包括忙等)时间是否达到 threshold_time。
3. 计算阈值threshold_time：threshold_time 必须大于进程持有锁的时间 lock_time，否则会将正常持有锁的进程判为异常进程。
4. 统计进程持有锁时间lock_time：加锁函数与解锁函数之间的时间便是线程持有该锁的时间。

Java死锁的分析与研究

- 获取锁🔒的等待者算法

1. 异常进程：等待时间超过阈值threshold_time的进程。
2. 计算等待时间：使用进程描述符 task_struct 结构体中的last_arrival 为参考，来判断进程的等待(包括忙等)时间是否达到 threshold_time。
3. 计算阈值threshold_time：threshold_time 必须大于进程持有锁的时间 lock_time，否则会将正常持有锁的进程判为异常进程。
4. 统计进程持有锁时间lock_time：加锁函数与解锁函数之间的时间便是线程持有该锁的时间。
5. 统计时间片time_slice:在进程的描述符中会记录进程总的运行时间(sum_exec_runtime)以及运行次数(pcount)，这两者之间的比值便是进程的平均时间片 time_slice。

- 筛选异常进程算法

1. 当进程执行加锁函数的时候，内核会将加锁函数的返回地址压入进程的内核栈。

- 筛选异常进程算法

1. 当进程执行加锁函数的时候，内核会将加锁函数的返回地址压入进程的内核栈。
2. 当进程从加锁函数成功退出的时候，再将该地址从栈中弹出。

- 筛选异常进程算法

1. 当进程执行加锁函数的时候，内核会将加锁函数的返回地址压入进程的内核栈。
2. 当进程从加锁函数成功退出的时候，再将该地址从栈中弹出。
3. 内核提供的函数 kallsyms_lookup 可以根据该地址找到对应的加锁函数。

- 筛选异常进程算法

1. 当进程执行加锁函数的时候，内核会将加锁函数的返回地址压入进程的内核栈。
2. 当进程从加锁函数成功退出的时候，再将该地址从栈中弹出。
3. 内核提供的函数 kallsyms_lookup 可以根据该地址找到对应的加锁函数。
4. 可以通过判断异常进程的内核栈上有无加锁函数，来从异常进程中筛选出锁的等待者。

Java死锁的分析与研究

- 循环等待图⏳

根据死锁检测算法，为了检查线程间是否发生了死锁，需要判断线程的资源依赖关系是否会形成一个循环等待图。

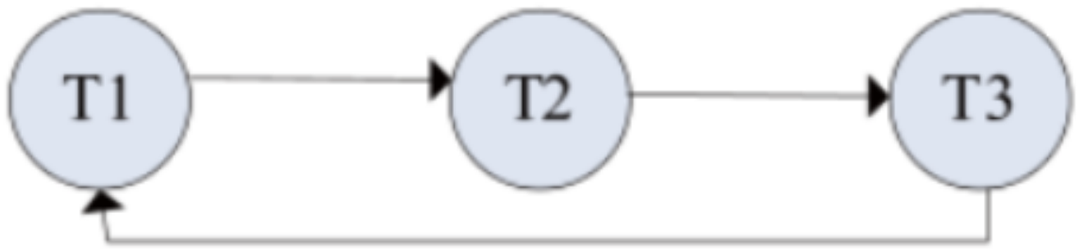


图7 死锁中的循环等待图

Java死锁的分析与研究

- 实验结果🧪

为了检测死锁检测方法的有效性，使用操作系统中的信号量模拟三个线程访问三种临界资源。

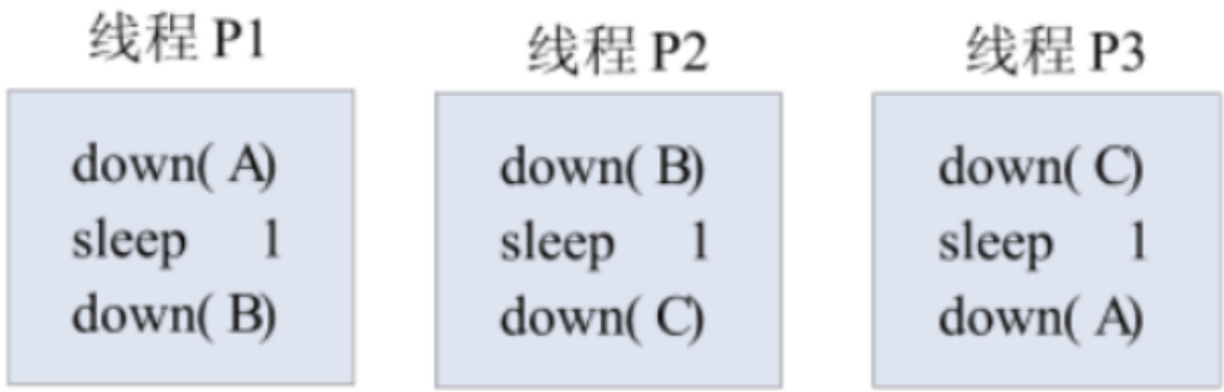


图8 三个线程产生的信号量死锁

Java死锁的分析与研究

●实验结果🧪

1. 临界资源使用资源地址来标识，进程 2471 拥有资源a0(资源地址前16位均一样，只使用后两位标识)，申请资源 c0。

```
1  pid 2471  acquire resource ffffffffa00903 a0
2  pid 2472  acquire resource ffffffffa00903 c0
3  pid 2473  acquire resource ffffffffa00903 e0
4  pid 2471  wait resource ffffffffa00903 c0
5  pid 2472  wait resource ffffffffa00903 e0
5  pid 2473  wait resource ffffffffa00903 a0
7  2471 ->2472 2472 ->2473 2473 ->2471
8  above process has dead lock
```

图9 三线程死锁检测结果

Java死锁的分析与研究

●实验结果🧪

1. 临界资源使用资源地址来标识，进程 2471 拥有资源a0(资源地址前16位均一样，只使用后两位标识)，申请资源 c0。
2. 进程2473拥有资源e0，申请资源a0。

```
1  pid 2471   acquire resource ffffffffa00903 a0
2  pid 2472   acquire resource ffffffffa00903 c0
3  pid 2473   acquire resource ffffffffa00903 e0
4  pid 2471  wait resource ffffffffa00903 c0
5  pid 2472  wait resource ffffffffa00903 e0
5  pid 2473  wait resource ffffffffa00903 a0
7  2471 ->2472 2472 ->2473 2473 ->2471
8  above process has dead lock
```

图9 三线程死锁检测结果

Java死锁的分析与研究

●实验结果🧪

1. 临界资源使用资源地址来标识，进程 2471 拥有资源a0(资源地址前16位均一样，只使用后两位标识)，申请资源 c0。
2. 进程2473拥有资源e0，申请资源a0。
3. 通过资源依赖的传递关系，可以得出进程 2471 指向自己的依赖关系：2471->2471。

```
1  pid 2471  acquire resource ffffffffa00903 a0
2  pid 2472  acquire resource ffffffffa00903 c0
3  pid 2473  acquire resource ffffffffa00903 e0
4  pid 2471  wait resource ffffffffa00903 c0
5  pid 2472  wait resource ffffffffa00903 e0
5  pid 2473  wait resource ffffffffa00903 a0
7  2471 ->2472 2472 ->2473 2473 ->2471
8  above process has dead lock
```

图9 三线程死锁检测结果

Java死锁的分析与研究

●实验结果🧪

1. 临界资源使用资源地址来标识，进程 2471 拥有资源a0(资源地址前16位均一样，只使用后两位标识)，申请资源 c0。
2. 进程2473拥有资源e0，申请资源a0。
3. 通过资源依赖的传递关系，可以得出进程 2471 指向自己的依赖关系：2471->2471。

=> 进程间存在存在死锁

```
1  pid 2471   acquire resource ffffffffa00903 a0
2  pid 2472   acquire resource ffffffffa00903 c0
3  pid 2473   acquire resource ffffffffa00903 e0
4  pid 2471  wait resource ffffffffa00903 c0
5  pid 2472  wait resource ffffffffa00903 e0
5  pid 2473  wait resource ffffffffa00903 a0
7  2471 ->2472 2472 ->2473 2473 ->2471
8  above process has dead lock
```

图9 三线程死锁检测结果

谢谢

第 9 组

2019.6.3