实战Java高并发程序设计第2周

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

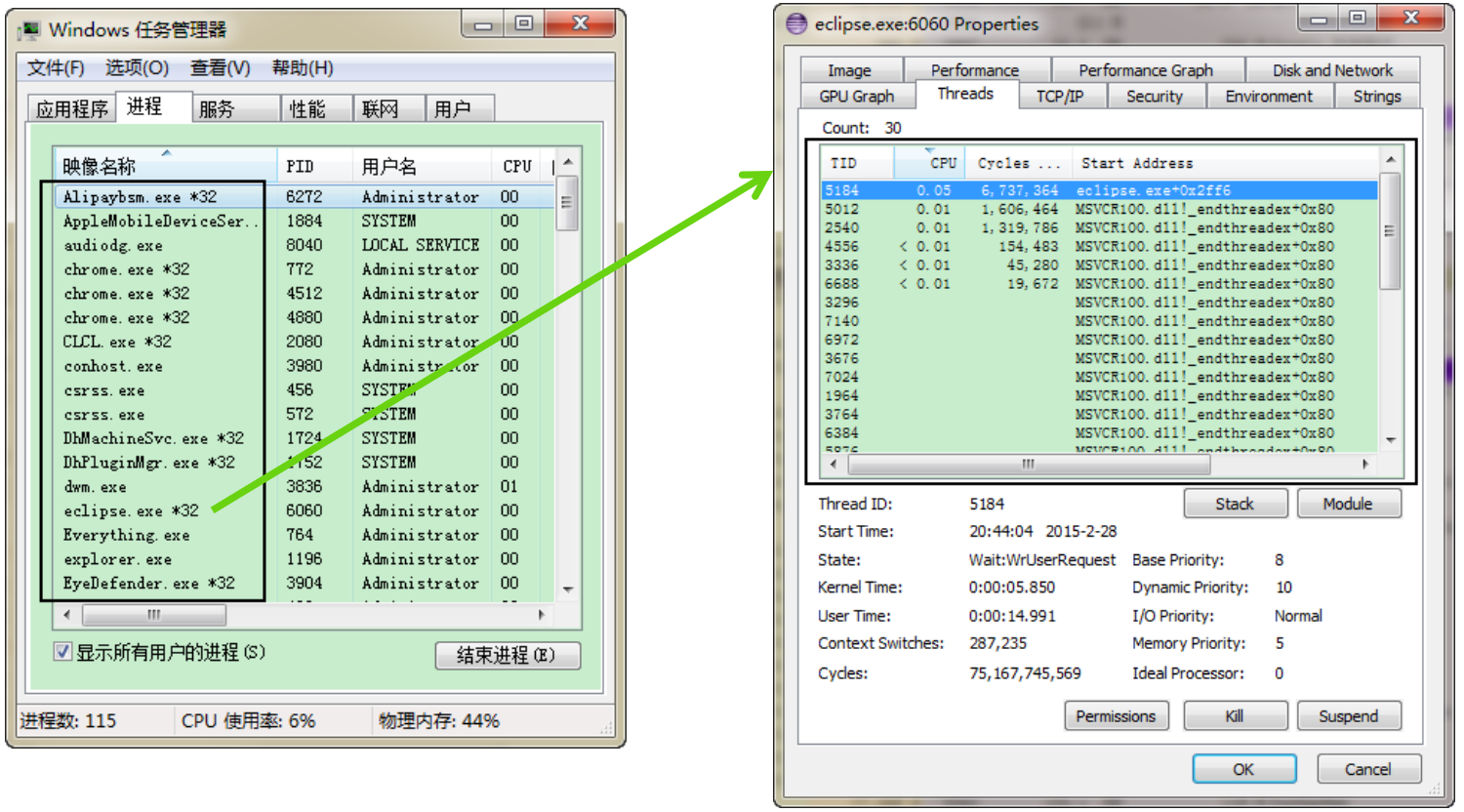
- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

内容提要

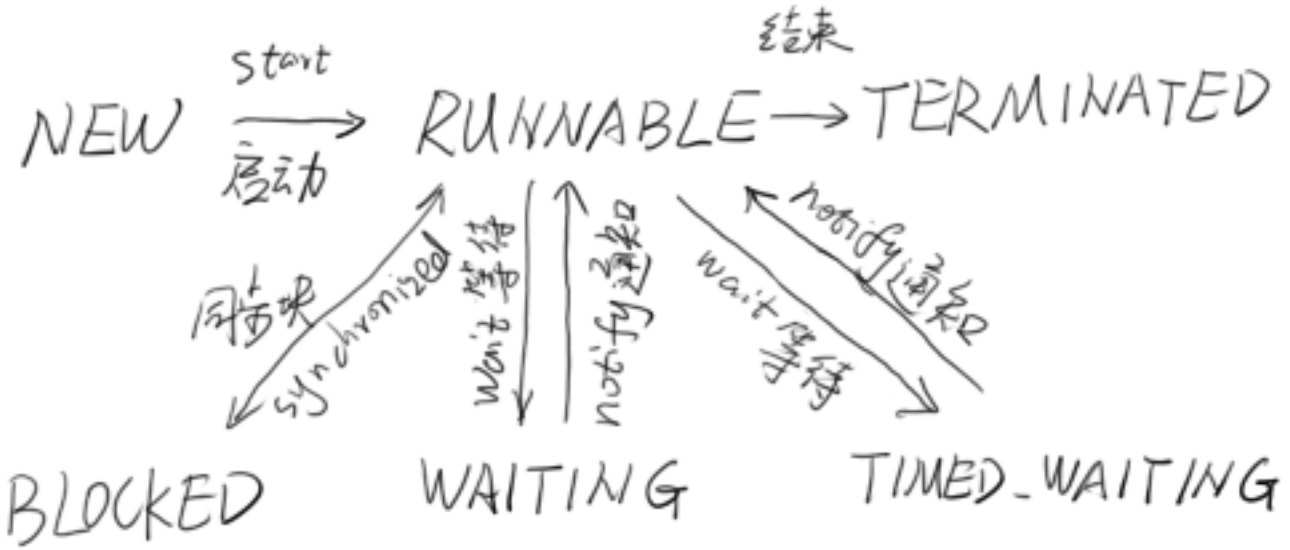
- 什么是线程
- 线程的基本操作
- 守护线程
- 线程优先级
- 基本的线程同步操作

什么是线程

- 什么是线程
  - 线程是进程内的执行单元



线程的基本操作


NEW
start
启动
RUNNABLE
结束
TERMINATED
同步块
synchronized
wait 等待
notify 通知
wait 等待
notify 通知
BLOCKED
WAITING
TIMED_WAITING

线程的基本操作

- 新建线程

```
Thread t1=new Thread();
t1.start();
```

```
Thread t1=new Thread();
t1.run(); //不能开启线程
```

```
Thread.run()的实现
target 是Runnable接口
public void run() {
    if (target != null) {
        target.run();
    }
}
```

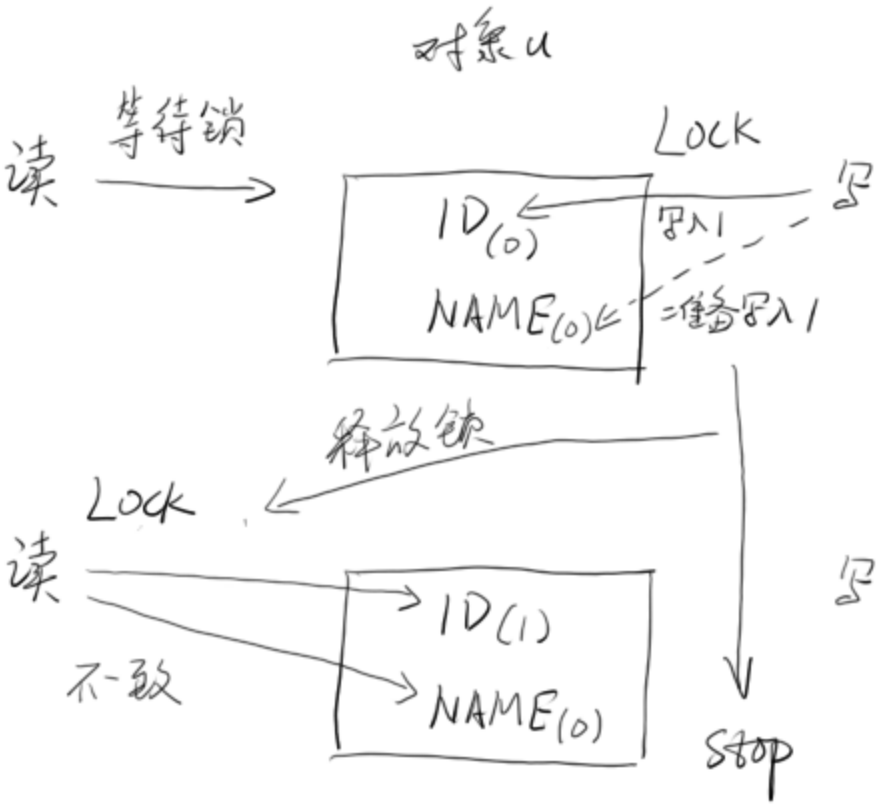
```
Thread t1=new Thread(){
            @Override
            public void run(){
                        System.out.println("Hello, I am t1");
            }
};
t1.start();
```

```
Thread t1=new Thread(new CreateThread3());
t1.start();
```

线程的基本操作

- 终止线程
  - Thread.stop() 不推荐使用。它会释放所有monitor

记录1：ID=1，NAME=小明
记录2：ID=2，NAME=小王



线程的基本操作

- 中断线程

```
public void Thread.interrupt()          // 中断线程
public boolean Thread.isInterrupted()   // 判断是否被中断
public static boolean Thread.interrupted() // 判断是否被中断，并清除当前中断状态
```

```
public void run(){
        while(true){
                Thread.yield();
        }
}

t1.interrupt();
```

```
public void run(){
        while(true){

        if(Thread.currentThread().isInterrupted()){

        System.out.println("Interruted!");
                        break;
        }
                Thread.yield();
        }
}
```

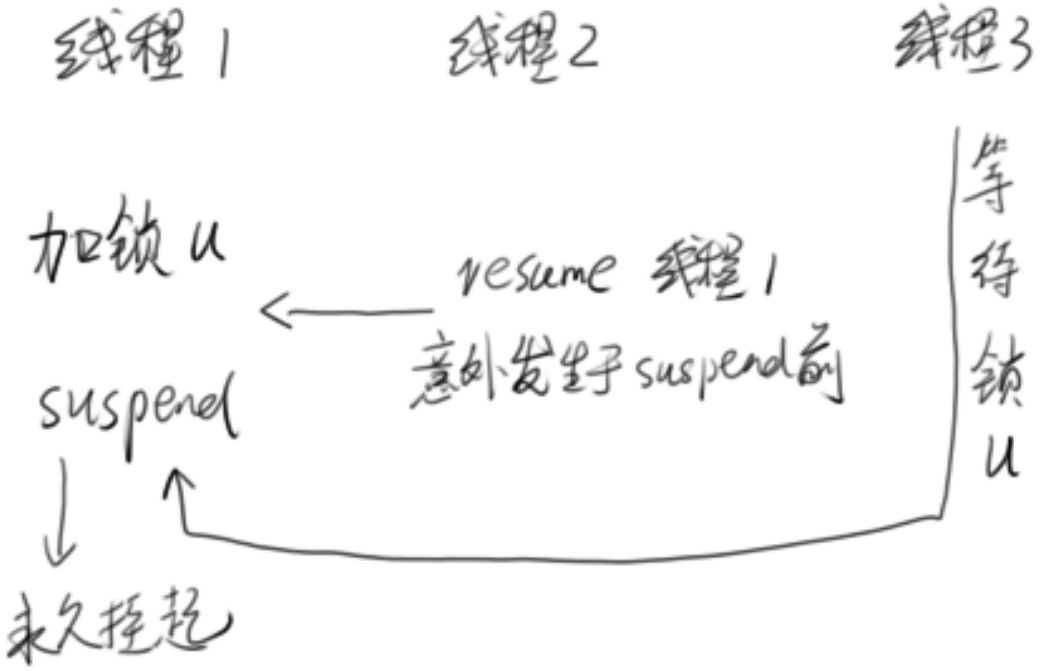
线程的基本操作-线程中断

- public static native void sleep(long millis) throws InterruptedException

```
public void run(){
        while(true){
                if(Thread.currentThread().isInterrupted()){
                        System.out.println("Interruted!");
                        break;
                }
                try {
                        Thread.sleep(2000);
                } catch (InterruptedException e) {
                        System.out.println("Interruted When Sleep");
                        //设置中断状态，抛出异常后会清除中断标记位
                        Thread.currentThread().interrupt();
                }
                Thread.yield();
        }
}
```

线程的基本操作

- 挂起（suspend）和继续执行（resume）线程
  - suspend()不会释放锁
  - 如果加锁发生在resume()之前 ，则死锁发生



线程的基本操作

- 等待线程结束（join）和谦让(yeild)

```
public final void join() throws InterruptedException
public final synchronized void join(long millis) throws InterruptedException
```

```
public class JoinMain {
    public volatile static int i=0;
    public static class AddThread extends Thread{
        @Override
        public void run() {
            for(i=0;i<10000000;i++);
        }
    }
    public static void main(String[] args) throws InterruptedException {
        AddThread at=new AddThread();
        at.start();
        at.join();
        System.out.println(i);
    }
}
```

```
join的本质
while (isAlive()) {
    wait(0);
}
```

线程执行完毕后，
系统会调用
notifyAll()

不要在Thread实例上使用 wait()和notify()方法

守护线程

- 在后台默默地完成一些系统性的服务，比如垃圾回收线程、JIT线程就可以理解为守护线程
- 当一个Java应用内，只有守护线程时，Java虚拟机就会自然退出

```
Thread t=new DaemonT();
t.setDaemon(true);
t.start();
```

线程优先级

- public final static int MIN_PRIORITY = 1;
- public final static int NORM_PRIORITY = 5;
- public final static int MAX_PRIORITY = 10;

```
Thread high=new HightPriority();
LowPriority low=new LowPriority();
high.setPriority(Thread.MAX_PRIORITY);
low.setPriority(Thread.MIN_PRIORITY);
low.start();
high.start();
```

高优先级的线程更容易再竞争中获胜

基本的线程同步操作

- synchronized
  - 指定加锁对象：对给定对象加锁，进入同步代码前要获得给定对象的锁。
  - 直接作用于实例方法：相当于对当前实例加锁，进入同步代码前要获得当前实例的锁。
  - 直接作用于静态方法：相当于对当前类加锁，进入同步代码前要获得当前类的锁。
- Object.wait() Obejct.notify()

基本的线程同步操作

指定加锁对象

```
public void run() {
        for(int j=0;j<10000000;j++){
                synchronized(instance){
                        i++;
                }
        }
}
```

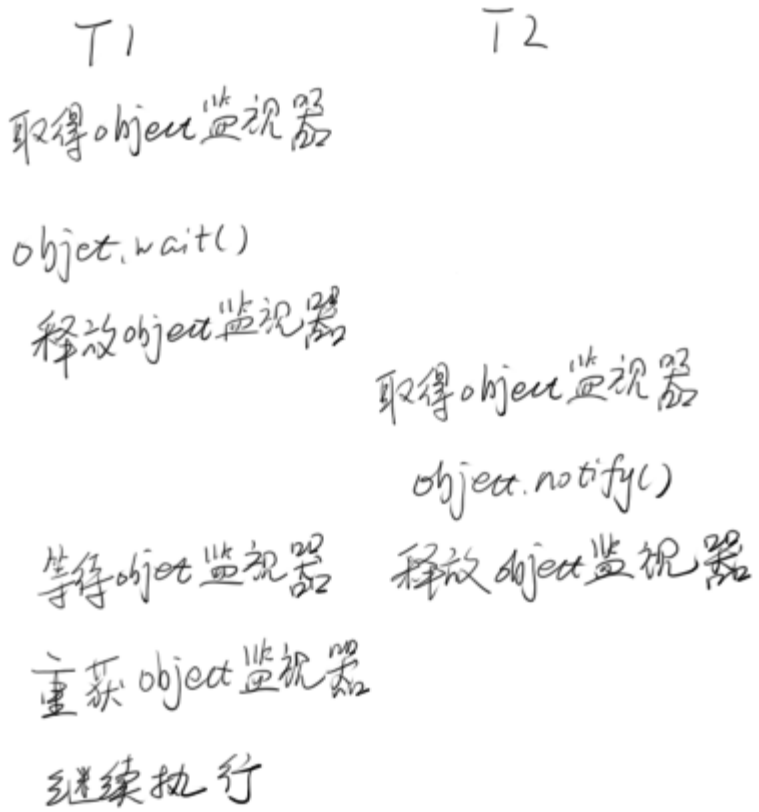
用在方法上

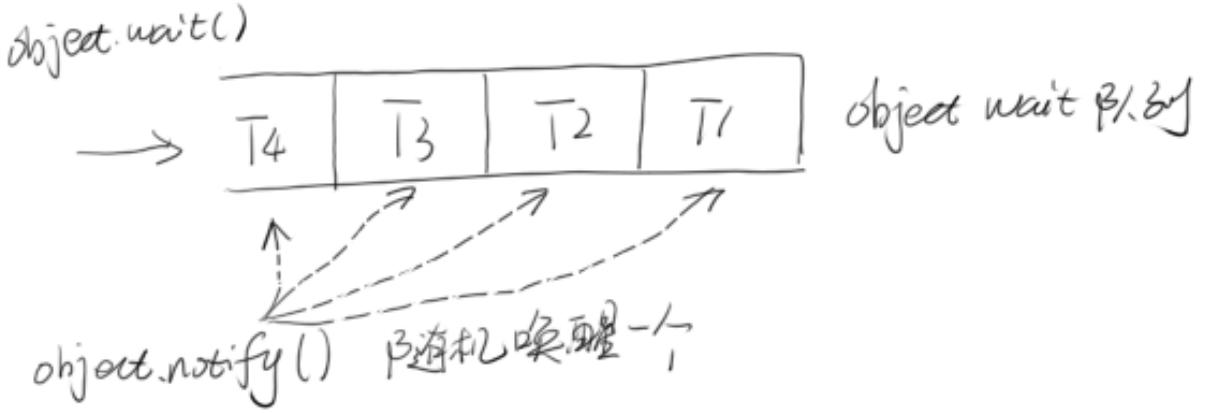
```
public synchronized void increase(){
        i++;
}
```

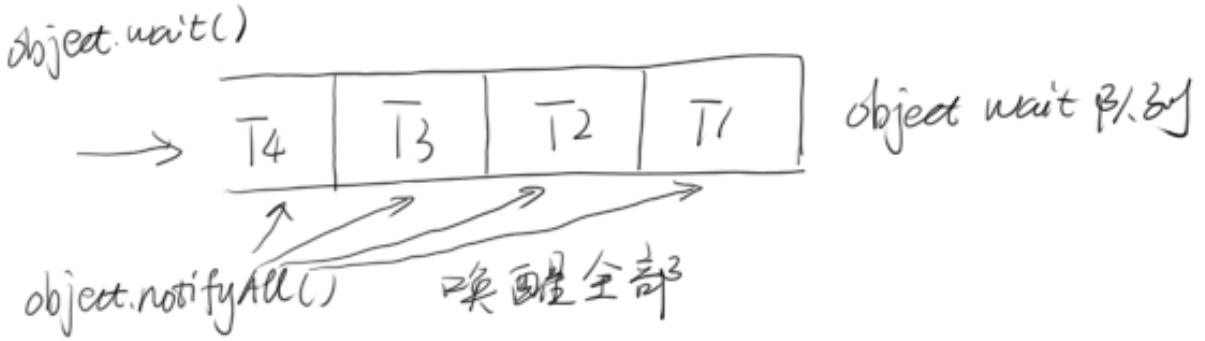
```
public static synchronized void increase(){
        i++;
}
```

基本的线程同步操作

- Object.wait() Obejct.notify()







基本的线程同步操作

- Object.wait() Obejct.notify()

  会释放锁

```
public static class T1 extends Thread{
    public void run()
    {
        synchronized (object) {
            System.out.println(System.currentTimeMillis()+":T1 start! ");
            try {
              System.out.println(System.currentTimeMillis()
                      +":T1 wait for object ");
               object.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(System.currentTimeMillis()+":T1 end!");
        }
    }
```

```
public static class T2 extends Thread{
    public void run()
    {
        synchronized (object) {
            System.out.println(System.currentTimeMillis()
                    +":T2 start! notify one thread");
            object.notify();
            System.out.println(System.currentTimeMillis()+":T2 end!");
            try {
              Thread.sleep(2000);
              } catch (InterruptedException e) {
            }
        }
    }
}
```

```
1425224592258:T1 start!
1425224592258:T1 wait for object
1425224592258:T2 start! notify one thread
1425224592258:T2 end!
1425224594258:T1 end!
```

Thanks

FAQ时间