京东一面：  
网络：三次握手四次挥手状态码，ddos, syn flood

答案：1、三次握手TCP建立连接，第一次客户端发送一个标志位SYN=1，seq=q的报文段给服务器端，请求建立连接，第二次握手，服务端收到报文后，通过标志位SYN=1知道，客户端发出建立连接的请求，然后服务端做出响应，将标志位SYN=1、ACK=1、ack=q+1、seq=p的报文发送给客户端，建立连接；第三次握手，客户端收到服务端建立连接的确认报文后，然后向服务端发送了一个标志位ACK=1，ack=p+1，seq=k的确认收到确认报文的报文，连接建立，并且第三次握手，实际上就可以发送数据了！2、为什么三次握手，目的是为了防止服务端产生多个无效的连接，占用资源，造成资源的浪费：因为假设第二次握手的确认信息由于网络等原因，没有到达客户端，客户端以为服务端没有建立连接，会继续发送建立连接的请求，而服务端则可能认为客户端又发出了连接建立的新的请求，会确认，这样导致服务端产生了多个无效的连接了！

数据库：索引类型，引擎类型，事务

答案：1、普通索引；唯一索引、主键索引、联合索引、全文索引 alter table tableName add index indexname(‘字段’) 或者create index indexname(‘字段’) on table tableName

1. INNODB和MYISAM
2. 事务ACID、隔离级别

算法：排序稳定性，红黑树，b-,b 树

设计模式：设计原则，几种设计模式

**操作系统：内存管理，系统调用**

java：锁

一面

求两个链表的交叉部分

**堆排序**

平衡树如何调整

项目讲一讲

Kafka讲一讲

Docker平时怎么使用的

**几种线程池区别**

Kafka如何解决数据堆积

二面

讲一讲java的集合

并发工具包

二叉树了解哪些

HashMap如何解决Hash冲突

阻塞队列有哪些

队列有哪几种消费方式

wait和await的区别

几万条数据的链表如何找到中间节点

常用的linux命令

aio，nio，bio的了解

三面

项目中的难点

mysql的存在插入不存在更新怎么实现的

如何保证线程安全

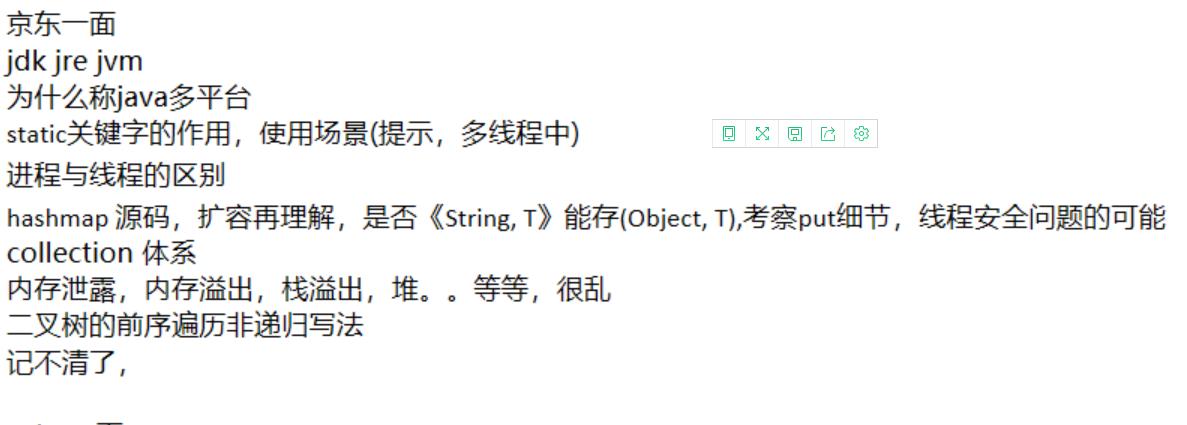
从二叉树中找到最大的二叉搜索子树

最后一个算法没写对，面试就结束了，最后约了hr面，估计还是凉了要。

京东一面（1个小时）：  
没有自我介绍，直接开始怼：  
1 项目相关，从前端到后端，到数据库，从单机到集群，到分布式解决高并发访问的各种方法细节  
2 分布式锁实现方式，三种方式，详细说下redis和zk的实现  
3 RBAC的内容  
4 分库分表的具体解决方案，水平切分时怎么选取切分规则，有哪些问题  
5 自己怎么实现一个类似于fastjson这样的json序列化方法  
6 syschronized和lock的区别和原理  
7 threadlocal原理及其应用  
8 一句话说明进程和线程，进程通信方式  
9 快速排序和堆排序对比，快排怎么优化  
10 jdbc操作数据库流程，如果加入事务，流程又是怎么样  
11 http协议中怎么区分头部和包体（居然是空格区分，一口老血）  
12 linux相关指令的问题

Scp远程下载文件 、 top指令 、 netstat -ano端口 、 ps -aux进程 、 d**rwx**r-xr-x 文件类型、当前用户权限、同一组的用户权限、其他用户的权限 4 2 1

京东二面（40分钟）：  
1说说你实习的项目吧，缓存策略怎么样的，批量查询阀值，缓存灰度占比，缓存击穿，缓存穿透，更新策略，埋点监控等等  
2高并发场景处理策略，单机，集群，分布式情况下的区别，数据库怎么优化  
4 并发包下的各种类，其中countdownlatch和回环栅栏实际场景中应该怎么应用  
5 CAP的理解和实际场景下一致性与可用性的选择  
6 你有什么想问的



一面：

0 自我介绍，讲一下项目中的多线程实现

1 Java的线程池的参数，拒绝策略，阻塞队列等实现和使用

2 fixethreadpool使用的是什么阻塞队列，如果使用arrayblockingqueue或者linkedblockingqueue会有什么问题。

3 ArrayList和linkedlist有什么区别，扩容呢

4 hashmap，hashtable，concurrenthashmap1.7和1.8选一个说一下。

5 jdk1.8针对hashmap使用红黑树优化目的是什么，红黑树的结构说一下。

**红黑树的特性**:

**（1）每个节点或者是黑色，或者是红色。**

**（2）根节点是黑色。**

**（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]**

**（4）如果一个节点是红色的，则它的子节点必须是黑色的。**

**（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。**

6 JVM了解么，说一下内存结构，堆区的分代，垃圾回收以及所用算法。

7 为什么要划分年轻代和老年代，方法区是否会垃圾回收，还问了gc时要扫描哪些位置，说了gc root，他问具体呢，虚拟机栈的局部变量，然后说其他忘了。

在Java虚拟机中判断一个对象是否可以被回收，有一种做法叫可达性分析算法，也就是从GC Root到各个对象，如果GC Root到某个对象还有可达的引用链，那么这个对象就还不能被回收，否则就等着被收割吧。

这里既然提到了GC Root，那么哪类对象可以作为GC Root呢，这是一个在面试中经常被问到的问题。

《深入理解Java虚拟机》一书中是这么说的，一下几种对象可以作为GC Root：

* 虚拟机栈中的引用对象
* 方法区中类静态属性引用的对象
* 方法区中常量引用对象
* 本地方法栈中JNI引用对象

8 MySQL了解么，说一下建立索引要考量的点，说了索引筛选率，过滤使用的字段，以及explain查看是否使用了索引等。

1、表的主键、外键必须有索引；

2、数据量超过300的表应该有索引；

3、经常与其他表进行连接的表，在连接字段上应该建立索引；

4、经常出现在Where子句中的字段，特别是大表的字段，应该建立索引；

5、索引应该建在选择性高的字段上；

6、索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引；

7、频繁进行数据操作的表，不要建立太多的索引；

9 MySQL的隔离级别有哪些，默认级别是什么，他说是可重复读。

10 为什么select方法默认可以不加锁呢，明明可能会有并发冲突，我说根据隔离级别判断是否加锁，他说可重复读级别下select是不加锁的，为什么。

我说应该是用了乐观锁，也就是MVCC多版本并发控制，适用于读多写少，所以不加锁。

11 Redis提供有哪些数据类型

二面：

1 讲一下两个项目，分别做了什么，觉得跟自己做的项目有什么不同，如何看待这种不同，应该如何改进。

2 平时怎么学习，你的博客一般写了什么，觉得和高质量博客的差距在哪里。

我说我原创的都是比较基础的，进阶一点的博客主要是整理别人的文章。

3 Java虚拟机了解么，讲一下内存分区和gc，如何排查堆内存的问题，说使用jmap转储dump文件分析或者使用jstat等工具检测gc。

4 Java的线程池了解么，看过它的源代码么，怎么修改源码可以实现线程状态的监控呢。

visualvm工具

5 web框架了解什么，讲讲SpringMVC的启动过程，讲了mvc上下文以及dispatcher初始化过程和请求流程。

6 设计模式了解么，单例，工厂，分别出现在什么场景。

Spring，servlet，Spring的工厂方法

7 MySQL的select1和select \*有什么区别，为什么加索引访问比不加索引要快。

8 负载均衡的几种算法，缓存的几种淘汰策略。

9 计算机网络了解么，http的header有哪些字段，是否包含ip地址。

10 有什么想问，能先来实习么，不能。

我说了不能先实习，然后就没有然后了，emmm，等了2周没有动静，估计凉凉。

**京东一面30分钟：**

1.介绍项目

2.hashmap，concurrenthashmap，效率

3.线程池，及如何设置线程池数量

4.对称加密，非对称加密

5. lock syn 以及效率

6. 设计模式原则

7.策略模式和命令模式

8. 数据库事物隔离

9.还有的不记得了。。。

京东二面30分钟左右，主要问项目，中间插了几个简单的基础

1. 项目，主要问项目，其它基本没问

2. 乐观锁 悲观锁

3. equals ＝＝

4. 多态

答案：1、编译时多态和运行时多态，编译时多态，主要表现形式是方法的重载，即在编译的时候，就可以确定是调用哪一个重载的方法；运行时多态，主要表现形式是方法的重写，即只有在运行的时候，才可以确定是调用具体哪一个方法 2、方法重载：发生在同一个类当中的，方法名相同，但是参数列表不同，返回类型可以相同或者不同，访问修饰符不做要求 ； 方法重写：是发生在子类和父类之间的，方法名，参数列表相同，返回类型都要相同，访问修饰符，子类的要大于等于父类的

1、用户表（userID、userName、userPhore、userAddress）；商品表（goodID、goodName、goodPrice、goodStock、startTime、endTime） ； 订单表（orderID、userID、goodID）

重写Equals方法

Public class People{

String name;

String phore;

get和set方法

Public boolean equals(Object obj){

If(obj != null && obj instanceOf People){

People p = (People)obj;

If(this.name.equals(p.name) && this.phore.equals(p.phore)){

Return true;

}

}

Return false;

}

}

快速排序

Public class Main{

Public static void main(String args[]){

Int[] arr = { 1, 2,3,4,5,12,23,1};

quickSort(arr,0,arr.length-1);

}

Public static void quickSort(int[] arr , int left , int right){

**If(left >= right){**

Return;

}

**Int base = division(arr,left,right);**

quickSort(arr,left,base-1);

quickSort(arr,base+1,right);

}

Public static int division(int[] arr , int left , int right){

Int value = arr[left];

**While(left < right){**

While(left<right && arr[right]>=value){

Right--;

}

Arr[left] = arr[right];

While(left < right && arr[left]<=value){

Left++;

}

Arr[right] = arr[left];

}

Arr[left] = value;

Return left;

}

}

归并排序

Public class Main{

Public static void main(String[] args){

}

Public static void mergeSort(int[] arr , int left , int right){

**If(left >= rught){**

Reutrn;

}

Int mid = left+(right-left)/2;

mergeSort(arr,left,mid);

mergeSort(arr,mid+1,right);

**Merge(arr,left,mid,right);**

}

Public static void merge(int[] arr , int left , int mid , int right){

Int[] temp = new int[right-left+1];

Int i = left ;

Int j = mid+1;

Int k = 0;

While(i<=mid && j <=right){

If(arr[i] < arr[j]){

Temp[k++] = arr[i++];

}else{

Temp[k++] = arr[j++];

}

}

While(i<=mid){

Temp[k++] = arr[i++];

}

While(j<=right){

Temp[k++] = arr[j++];

}

For(int i = 0 ; i < temp.length ; i++){

Arr[left+i] = temp[i];

}

}

}

堆排序

Public class Main{

Public static void main(String[] args){

}

Public static void heapSort(){

For(int i = arr.length/2 ; i>=0 ;i--){

Adjustment(arr,i,arr.length-1);

}

For(int i = arr.length-1 ; i>=0 ;i--){

Int temp = arr[i];

Arr[i] = arr[0];

Arr[0] = temp;

Adjustment(arr,0,i-1);

}

}

Public static void adjustment(int[] arr , int pos , int len){

Int value = arr[pos];

For(int i = 2\*pos+1 ; i < len ; i = i\*2+1){

If(i<len && arr[i]<arr[i+1]){

I++;

}

If(value > arr[i])

Break;

Arr[pos] = arr[i];

Pos = i;

}

Arr[pos] = value;

}

}

1:生产者消费者问题实现

第一种：semaphore信号量实现**(acquire获取许可，release释放许可**）

Semaphore mutex = new Semaphore(1);

Semaphore empty = new Semaphore(20);

Semaphore full = new Semaphore(0);

生产者：

**Mutex.acquire();**

**Empty.acquire();**

**生产。。。。**

**Mutex.release();**

**full.release();**

**第二种：使用synchronized+wait+notify+queue实现**

Class Hander{

Private int capacity = 100;

Private LinkedList<Object> queue = new LiskedList<>();

Public void produce(){

Synchronized(queue){ //加锁

While(queue.size()>=capacity){

Ssytem.out.println(“已经满了!”);

Queue.wait();//释放锁

}

Queue.addLast(new Object());

Queue.notifyAll();

}

}

Public void consume(){

Synchronized(queue){

While(queue.size()==0){

Ssytem.out.println(“已经空了!”);

Queue.wait();

}

Queue.removeFirst();

Queue.notifyAll();

}

}

}

Public class Main{

Public sattic void main(String args[]){

Hander hander = new Hander();

New Thread(new Runnable(){

Public void run(){

While(true){

Hander.produce();

}

}

}).satrt();

New Thread(new Runnable(){

Public void run(){

While(true){

Hander.consume();

}

}

}).satrt();

}

}

**第三种：使用reentrantLock+fulllock+emptyLock+queue实现**

Class Hander{

Private int capacity = 100;

Private LinkedList<Object> queue = new LinkedList<>();

Private ReentrantLock lock = new ReentrantLock();

Private Condition empty = lock.newCondition();

Private Condition full = lock.newCondition();

Public void produce(){

Lock.lock();

While(queue.size()>=capacity){

System.out.println(“已经满了”);

Full.await();

}

Queue.addLast(new Object());

Empty.singalAll();

Lock.unlock();

}

Public void consume(){

Lock.lock();

While(queue.size() == 0){

System.out.println(“已经空了”)；

Empty.await();

}

Queue.removeFirst();

Full.singalAll();

Lock.unlock();

}

}

Public class Main{

Public static void main(String args[]){

Hander hander = new Hander();

New Thread(new Runnable(){

Public void run(){

While(true){

Hander.produce();

}

}

}).start();

New Thread(new Runnable(){}).start();

}

}

1. Java优点

答案：平台无关性、垃圾回收机制、异常处理机制

1. Java特性

封装（隐藏内部的具体实现，起到保护数据的作用）继承（子类可以继承父类中的非私有的变量和方法，可以提高代码的重用性）多态（编译时多态，即重载，运行时多态，如重写）

1. 抽象类和接口

抽象类中可以有**main方法和构造方法**，接口中不能有；

抽象类中可以有**普通的成员变量**，而接口中只能有public static final修饰的常量；

抽象类中可以有**具体的方法，抽象的方法**，接口中只能有抽象的方法；

抽象类中方法的**修饰符可以有public protected default**，而接口中只能输public

抽象类和接口都不能被实例化

1. Java反射

JVM在运行的时候才**动态地加载这个类**，调用对应的方法或者访问对应的属性，比如类名.class.getMethod()或者getField等等 ， 应用比如IOC容器、JDK动态代理、加载mysql驱动

1. equals和hashCode

答案：hashCode是根据“对象的内存地址来计算对应的hashCode值的，不同的对象，其hashCode肯定是不同的” ； Object中的equals方法“是比较两个对象的内存地址是否相同的” ； Java中规定：equals相同，其hashCode一定相同，equals不同，则hashCode可能相同可能不同 ，重写equals方法必须要重写hashcode方法，比如String比如hashSet

1. Final

修饰基本数据类型，表示的是一个常量，值不能改变，修饰引用数据类型，表示该引用不能再指向其他对象实例 ； 修饰方法，表示该方法不能被重写，修饰类表示该类不能够被继承 ； 修饰方法里面的参数，表示该变量只能读 不能写 ； final还可以实现“不可变类”

1. Static

修饰变量，表示类变量或者全局变量，可以通过类名直接访问 ； 修饰方法，成为类方法，可以通过类名直接访问 ； 修饰代码块，静态代码块，可以用于一些初始化工作 ； 修饰内部类，静态内部来，比如其中一个应用就是静态内部类实现单例模式 （static的main方法 ， static的单例模式）

1. String和StringBuilder和StringBuffer

String不可变类，线程安全的，是由final修饰的，内部是由private final char[] 数组来存储字符串对象的

StringBuilder是可变类，内容可变，是线程非安全的，运行效率较高

StringBuffer是可变类，内容可变，线程安全的，通过synchronized关键字实现线程安全的

1. String中+是如何实现的

**// 程序编译期即加载完成对象s1为"ab"**

String s1 = "a" + "b";

**// 这种方式，JVM会先创建一个StringBuilder，然后通过其append方法完成**

String s1 = "a";

String s2 = "b";

String s3 = s1 + s2;

// 等效于 String s3 = (new StringBuilder(s1)).**append(s2).toString();**

1. map、list、set

List：是有序的，允许有重复的元素，该接口具体的实现类有ArrayList动态数组，随机访问、LinkedList双向循环链表，继承了list接口和Dqueue接口，可以实现stack和queue功能、Vector集合线程安全的

Set：是无序的，不允许有重复元素的，具体实现类有hashSet（基于hashMap实现的）、treeSet（基于treeMap实现的）、LinkedHashSet（基于LinkedHashMap实现的）

Map：是存储key-value的键值对，具体实现类有：hashMap（数组+链表+红黑树）、hshTable（线程安全的、11、2倍+1、装载因子0.75）、treeMap（基于红黑树实现的有序的）、linkedhashMap（在hashMap基础上，加了一个双向链表，来记录元素的插入顺序）

1. hashSet如何保证 元素不重复的

答案：通过hashCode和equals方法，先计算对象的hashcode值，如果发生冲突，则在比较两个对象的

1. 死锁和死锁的原因

两个或者两个以上的线程因为竞争资源而造成的一种僵局，若没有外界作用，它们会一直这样僵持下去，造成死锁！

Java代码死锁：

Object obj01 = new Object();

Object obj02 = new Object();

Public void thread01Method(){

**Synchronized(obj01){**

System.out.println(“获取obj01的锁”);

**Synchronized(obj02){**

System.out.println(“获取obj02的锁”);

}

}

}

Public void thread02Method(){

**Synchronized(obj02){**

System.out.println(“获取obj02的锁”);

**Synchronized(obj01){**

System.out.println(“获取obj01的锁”);

}

}

}

产生死锁的四个必要条件：互斥条件、不可剥夺条件，请求与保持条件、环路等待条件

死锁的预防（破坏四个必要条件）死锁的避免（银行家算法）死锁的检测（简化资源分配图）死锁的解除（剥夺死锁进程资源或者杀死死锁的进程）

1. wait和sleep（**wait是final修饰的，不能被重写，sleep是本地方法**）

Wait是object类的方法，进程会释放锁，sleep是thread的方法，不会释放锁，会释放cpu

1. HashMap

原理：基于数组+链表+红黑树实现的，

hashmap的扩容问题：负载因子是0.75，当元素个数大于等于表的长度的0.75的时候，进行扩容，以2倍方式扩容，

为什么HashMap的初始容量会是16，为什么是2倍扩容：两个原因，第一个原因是hashMap中采用了hashcode(key)&table.length-1运算来计算entry对象的散列位置的，计算效率高于%取模运算，此外可以使得元素更佳分散，降低了发生冲突的可能，即方便通过&代替%运算，提高运算效率，此外使得分散，降低了冲突的可能性

实现简单的 get/put操作：get，先判断key是否为null，然后通过两次hash计算key的hashcode，找到对应的散列位置，如果为空，则返回null，否则比较当前key的equals方法，相同则返回对应的value ； put也是null、ahshcode、equals；

处理哈希冲突用的哪种方法（拉链），还知道什么处理哈希冲突的方法（开放地址检测），开放地址检测怎么实现的：拉链法、线性探测法、平方探测法

1. 线程Thread的状态、方法

答案：新建、就绪、运行、堵塞、死亡 ； start() sleep() yeild() join()

1. 线程创建的方式

继承Thread类、实现Runnable接口 run方法、实现callable接口 call方法，具有返回值、通过ThreadPool的executorService

1. Java中实现线程安全策略

答案：使用synchronized的同步锁实现的同步方法和同步代码块 ； 或者使用可重入锁reentrantLock ； 使用同步集合，比如vector、hashTable ；使用juc包下面 的一些并发集合并发类，比如Atomic原子类，concurrentHashMap、copyOnWriteArrayList ；还有ThreadLocal本地变量

18、Synchronized和lock区别

答案：synchronized，是基于JVM实现的锁，悲观锁，具体应用比如同步方法，同步代码块，当synchronized修饰非static方法时，锁住的是当前的对象，而当synchronized修饰static方法时，类似一个全局锁，锁住的是所有对象 ；

Lock，是一个接口，具体实现类有reentrantLock，是基于Java语言实现的，乐观锁，其构造方法可以指定公平锁还是非公平锁，其方法tryLock可以提供定时锁的功能，其lockInterrupbly可以实现中断锁的功能，并且需要手动去释放锁unlock

1. Java中悲观锁和乐观锁（CAS机制），mysql中悲观锁（select for update 和）乐观锁（version）
2. JUC包下的Semaphore、countDownLatch、CyclicBarrer

Semaphore信号量，可以用于控制同时并发的线程的数量，有acquire、release

countDownLatch计数器，可以让一个线程或者多个线程一直等待，直到其他线程执行完之后，在执行，通过减计数的方式，当计数器减为0的时候，所有等待的线程会被激活，开始运行，await和countDown两个方法

CyclicBarrier计数器，让一组线程相互等待，直到它们都达到某个状态的时候，所有线程才开始运行，是通过加计数的方式，当计数器等于给定的值的时候，所有等待的线程会被激活运行，计数器可以反复使用，只要重置即可

1. 线程池ThreadPool

作用：可以通过重用线程，减少线程创建和销毁的次数，提高资源的利用率，此外线程池还可以提供一些定时任务的执行等功能

创建：ExecutorService threadPool = Executors.newCacheThreadPool();

newCacheThreadPool newFixedThreadPool newScheduleThreadPool newSingleThreadExecutor

执行：execute(new runnable(){}) 、 submit

获取结果：Future或者FutureTask

关闭：shutDown或者shutDownNow

构建一个线程池的要求：需要存放任务的容器，比如堵塞队列 ； 需要存放线程的容器，比如使用

corePoolSize maxPoolSize keepAliveTime unit workQueue ThreadFactory rejecthander

**当时IO密集型的任务时，可以设置线程数为cpu核数\*2+1 ； 当为CPU密集型的任务时，线程数可以设置为cpu核数+1**

**拒绝策略：直接丢弃，替换到叫老的任务 ，抛出异常 ， 使用主线程去执行该任务**

当线程池的线程数大于corePoolSize的时候，如果有线程的空闲时间超过了keepAliveTime，那么会销毁该线程

corePoolSize -- 任务队列 -- maxPoolSize

CachedThreadPool()

线程数无限制

**有空闲线程则复用空闲线程，若无空闲线程则新建线程**

创建方法：

ExecutorService ThreadPool = Executors.newCachedThreadPool();

源码：

public static ExecutorService newCachedThreadPool() {

**return new ThreadPoolExecutor**(0, Integer.MAX\_VALUE,

60L, TimeUnit.SECONDS,

new SynchronousQueue<Runnable>());

}

FixedThreadPool()

定长线程池：

**可控制线程最大并发数（同时执行的线程数）**

超出的线程会在队列中等待

创建方法：

ExecutorService ThreadPool = Executors.newFixedThreadPool(100);

public static ExecutorService newFixedThreadPool(int nThreads) {

return new ThreadPoolExecutor(**nThreads, nThreads**, 0L, TimeUnit.MILLISECONDS, **new LinkedBlockingQueue<Runnable>()**);

}

NewcahceThreadPool：0 Integer.MAX 60l secnods synchronizeDQueue,即线程的数量几乎没有限制，当有空闲线程时，则重用，没有的时候，则创建新的线程去执行任务 ， 当线程空闲时间超过1分钟是，会销毁

<https://www.nowcoder.com/discuss/76933>

<https://blog.csdn.net/heyewu4107/article/details/71009712>

<http://www.mamicode.com/info-detail-2383504.html>

<https://www.west.cn/info/html/chengxusheji/Javajishu/20180723/4370208.html>