
1.

- 升 CPU 利 : 台主 上 会 个CPU , 们可以创 个 , 上 作 可以 个 分 不同 CPU去 , 个CPU 个 , 了 CPU 使 , 使 单 只 个CPU 使
- 们在 上 , 为了 升响 , 分, 减 , 单 些 作, 可以 分利 业务 , 会 串 业务 , 发 吻合 业务 分
- 单 :
  - 充分利 CPU 力;
  - 便 业务 分, 升

2.

- 例 : 下 分 发 信

3.

- 发 为了 , , 但 发 不 , 且 发 可 会 到 , : 内 上下 切 全

4.

- 原　：原，即 个不可再 分割　　原　　个　个 作 么全　　功 么全
- 可　：　个　　共享变　修 ,另 个　　刻 到 (synchronized,volatile)
- 　　：　　代　先后　　(　器可 会　令　　)

5. Java

- 出　　全　　原因　　三个原因:
  - 　切　　原　　决办 ：使　　之 同 synchronized 使　(lock)
  - 　　可　　决办 ：synchronized volatile LOCK，可以 决可
  - 　优化　　决办 ：Happens-Before 则可以 决

6.

- 发:　个任务在同 个 CPU 上，　分　　(交)　，从　上　　些任务同
- 　：单位　内，　个　器　　器同　　个任务，　　义上 “同　　”
- 串 ：　n个任务，　个　　于任务　　在 个　　以不 在　不全 况，也 不 在临 区

- 发 = 个人　台
- 　= 个人分 了 台
- 串 = 个人　使　台

7.

- 　：　　中包含 个　　，即在 个　中可以同　　个不同不同 任务

8.

- 可以　CPU 利　在　　中，　个　　候，CPU 可以　其不　，　　了　　也　允 单个　创　个各　任务

9.

- 　也　，以　　占 内 ，　　占 内 也　；
- 　　协 和　，以　CPU　　；
- 　之　共享　　会 互 响，　　决　共享

10.

- 什么　　和　?
  -

个在内 中 个 在 上 个

- 中 个 任务（ 制单元）， 在

一个进程至少有一个线程，一个进程可以运行多个线程，多个线程可共享数据。

- 与 区别
  - 区别: 作 分 单位， 器任务 和 单位
  - : 个 代 和 （ 上下 ）， 之 切 会 ； 可以 做 ，同 共享代 和 ， 个 和 器（PC）， 之 切
  - 包含关 : 个 内 个 ，则 不 ， （ ）共同 ； 分， 以 也 为
  - 内 分 : 同 共享 地址 和 ， 与 之 地址 和 互
  - 响关 : 个 后，在保 下不会 其他 产 响，但 个 可 个 以 健
  - : 个 入口 列和 出口 但 不 ， 依 在 中， 供 个 制，两 均可 发

11. ?

- 中 个 于 CPU 个 ， 个 CPU 在任 刻只 个 使 ，为了 些 到 ，CPU 取 为 个 分 个 候 会 于 其他 使 ， 个 于 上下 切
- : 前任务在 CPU 切 到另 个任务之前会先保 ，以便下 再切 回 个任务 ，可以再加 个任务 任务从保 到再加 上下切
- 上下 切 也 ， 可 器 ，在 几十上 切 中， 切 以，上下 切 味 CPU ，事 上，可 作 中 作
- Linux 与其他 作 （包 其他 Unix ） 优 ，其中 ，其上下 切 和 切

12.

- (User) : 在前台， 具体 任务， 主
- (Daemon) : 在后台，为其他前台 务 也可以 JVM 中 “ 人” ， 会 JVM 作

13. Windows Linux cpu

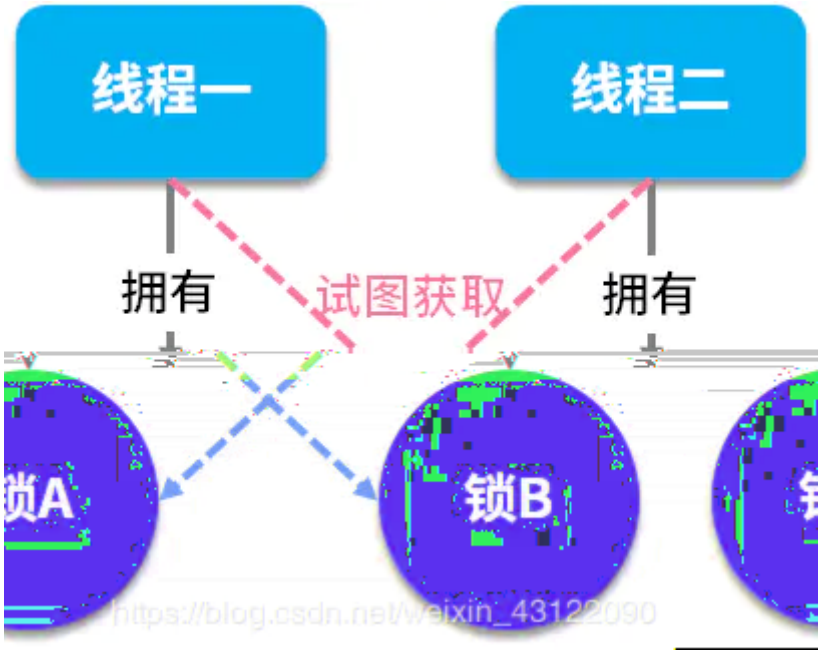
- windows上 任务 器 ，linux下可以 top 个 具
  - 出cpu 厉 pid， top命令， 后 下shift+p (shift+m 出 内) 出cpu利 厉 pid号

- 上 到 pid号，top -H -p pid 后 下shift+p， 出cpu利 厉
  号， top -H -p 1328
  - 取到 号 16 制，去 下
- 使 jstack 具 信 印 出，jstack pid号 > /tmp/t.dat， jstack 31365 > /tmp/t.dat
  - /tmp/t.dat 件， 号 信

或者直接使用JDK自带的工具查看“jconsole”、“visualVm”，这都是JDK自带的，可以直接在JDK的bin目录下找到直接使用

14.

- 两个 两个以上 （ ）在 中， 于 争 于 信
  ， 力作 ， 们 下去 于 产 了
  ， 些 在互 （ ） 为 （ ）
- 个 同 ， 们中 个 全 在 个 于 地
  ，因 不可
- 下图 ， A 2， B 1，他们同 ， 以 两个
  会互 入



15.

- 互 件：在 内 只 个 占 其 ， 只
  ， 占
- 占 且 件： 保 个 ，但又 出了 ， 其
  占 ， ，但又 其 保 不
- 不可 占 件：别人 占 了 ，你不 因为 也 ， 去 别人

- 件： 之 关 （ 个 合，A在
  B，B在 C，C在 A)

16.

1. 免 个 同 个
2. 免 个 在 内同 占 个 , 保 个 只占 个
3. 使 , 使 lock.tryLock(timeout) 代使 内 制

17.

- Thread ;

```
  public class MyThread extends Thread {
  @Override
  public void run() {
      System.out.println(Thread.currentThread().getName() + " run()方法正在执
行...");
  }
```

- Runnable 口;

```
  public class MyRunnable implements Runnable {
  @Override
  public void run() {
      System.out.println(Thread.currentThread().getName() + " run()方法执行
中...");
  }
```

- Callable 口;

```
  public class MyCallable implements Callable<Integer> {
  @Override
  public Integer call() {
      System.out.println(Thread.currentThread().getName() + " call()方法执行
中...");
      return 1;
  }
```

- 使 匿名内

```
  public class CreateRunnable {
      public static void main(String[] args) {
          //创建多线程创建开始
Thread thread = new Thread(new Runnable() {
              public void run() {
                  for (int i = 0; i < 10; i++) {
                      System.out.println("i:" + i);
                  }
              }
          });
          thread.start();
      }
  }
```

18. runnable callable

- 口
- 可以 写
- Thread.start()启动

- Runnable 口 run 回值；Callable 口 call 回值， 个 ，和Future FutureTask 合可以 取
- Runnable 口 run 只 出 ，且 ；Callable 口 call 允 出 ，可以 取 信 ：Callalbe 口 回 ， FutureTask.get() 到， 会 主 下 ， 不 不会

19. run() start()

- 个 个 Thread run() 其 作 ，run() 为 体 Thread start() 启动 个
- start() 于启动 ，run() 于 代 run() 可以 ， start() 只
- start() 启动 个 ， 了 start() run 体代 ，可以 其他 代 ； 于 ， 后 Thread run() 其 ，run() ， 后CPU再 其
- run() 在 ，只 个函 ， 不 run()，其 于 了 个 函 ， run() run() 下 代 ， 以 只 ， ， 以在 使 start() 不 run()

20. start() run() run()

另 个 典 java ， 且在 中会 到 单，但 人 会 不上 ！

- new 个Thread， 入了 start() ，会启动 个 使 入了 ， 分 到时间片后 可以 了 start() 会 准 作， 后 动 run() 内 ， 作
- run() ，会 run 个main 下 去 ， 不会在 个 中 ， 以 不 作

: start 可启动 使 入 ， run 只 thread 个 ， 在主

21. Callable Future?

- Callable 口 似于Runnable，从名 可以 出 了，但 Runnable不会 回 ， 且 出 回 ， Callable功 些， 后，可以 回值， 个 回值

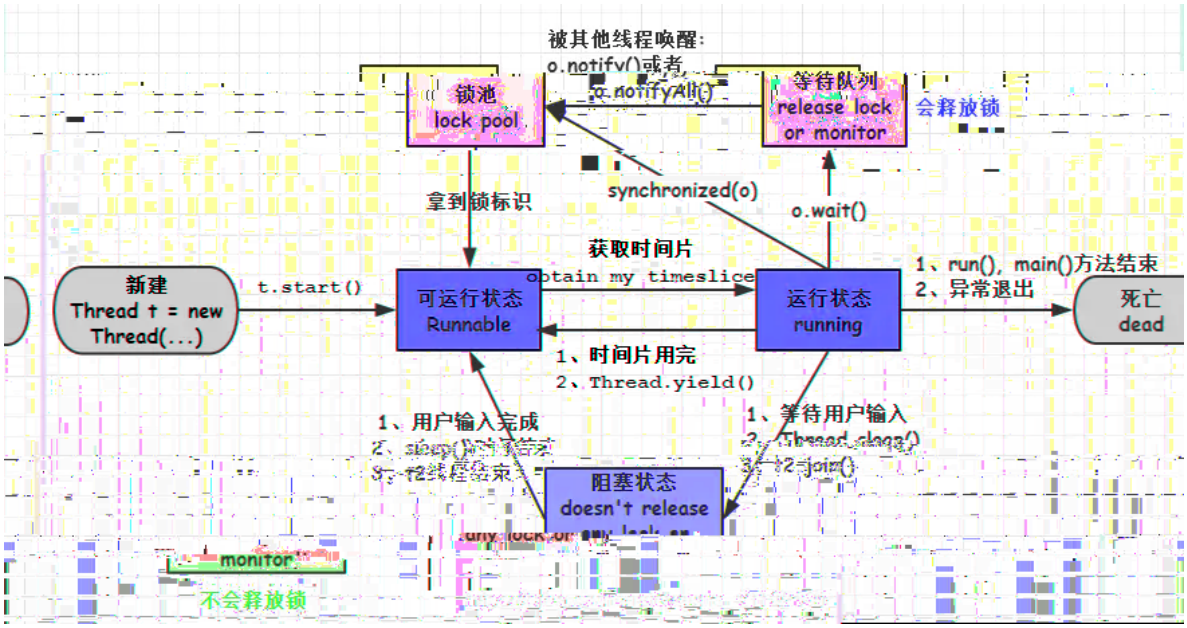
可以 Future 到，也 ，Future 可以 到 任务 回值

- Future 口 任务， 个可 任务 以 Callable 于产 ，Future 于 取

22. FutureTask

- FutureTask 个 任务 FutureTask 可以传入 个 Callable 具体 ，可以 个 任务 取 判 否 取 任务 作 只 候 取回， get 会 个 FutureTask 可以 了 Callable 和 Runnable 包 ， 于 FutureTask 也 Runnable 口 ，以 FutureTask 也可以 入 中

23.



- (new): 创 了 个
- (可) (runnable): 创 后， start() ， 于 ， 中， 取cpu 使
- (running): 可 (runnable) 了cpu (timeslice)， 代 : 入到 唯 入口，也 ， 入 ， 先 于 中;
- (block): 于 中 于 原因， CPU 使 ，停 ，入 ， 到其 入到 ， 会再 CPU 以 入到
  - 况分三 :
  - ( ). : 中 wait() ，JVM会 入 列(waitting queue)中，使 入到 ;
  - (二). 同 : 在 取 synchronized 同 (因为 其 占)，，则JVM会 入 (lock pool)中， 会 入同 ;
  - (三). 其他 : sleep() join() 发出了 I/O ， 会 入到 sleep() join() I/O ， 入
- 亡(dead)( ): run() main() ， 因 出了run() ，则 命周 亡 不可再

24. Java

- 只 个CPU，在任 刻只 器 令， 个 只 CPU 使 令 发 ，其 从 上 ，各个 CPU 使 ，分别 各 任务 在 中，会 个 于 在 CPU，JAVA 任务 ， 制为 个 分 CPU 使 (Java JVM中 器)
- 两 ：分 和 占
  - 分 cpu 使 ， 且 均分 个 占 CPU 个也
  - Java 占 ， 优先 可 中优先 占 CPU， 可 中 优先 同， 么 个 ，使其占 CPU 于 会 ， 不 不 CPU

25.

线程调度器选择优先级最高的线程运行，但是，如果发生以下情况，就会终止线程的运行：

- (1) 体中 了yield 出了 cpu 占 利
- (2) 体中 了sleep 使 入
- (3) 于IO 作受到
- (4) 另 个 优先 出
- (5) 在 中，

26. (Thread Scheduler) (Time Slicing)

- 器 个 作 务， 为Runnable 分 CPU 们创 个 启动 ， 便依 于 器
- 分 可 CPU 分 可 Runnable 分 CPU 可以 于 优先
- 不受到Java 制， 以 制 (也 不 你 依 于 优先)

27.

- (1) wait()：使 个 于 ( ) ， 且 ；
- (2) sleep()：使 个 在 于 ， 个 ， InterruptedException ；
- (3) notify()：唤 个 于 ， 在 候， 不 切 唤 个 ， JVM 唤 哪个 ， 且与优先 关；
- (4) notityAll()：唤 于 ， 不 ， 们 争，只 入 ；

28. sleep() wait()

两者都可以暂停线程的执行

- 不同：sleep() Thread ，wait() Object
- 否 ：sleep() 不 ；wait()
- 不同：Wait 于 交互/ 信，sleep 于 停
- 不同：wait() 后， 不会 动 ， 别 同 个 上 notify() notifyAll() sleep() 后， 会 动 可以使 wait(long timeout) 后 会 动

29. wait() if

- 于 可 会 到 和伪唤 ， 不在 中 件， 会在 件 况下 出
- wait() 在 ，因为 取到 CPU 候，其他 件可 ， 以在 前， 件 否 会 下 准 使 wait 和 notify 代 ：

```
synchronized (monitor) {
    //  判断条件谓词是否得到满足
  while(!locked) {
        //  等待唤醒
  monitor.wait();
    }
    //  处理其他的业务逻辑
}
```

30. wait(), notify() notifyAll() Object

- 因为Java 了Object，Java 任何 可以作为 ， 且 wait()，notify() 于 唤 ，在 Java 中 可供任何 使 ， 以任 义在Object 中
- 人会 ， ， 也可以 wait() 义在Thread 啊， 义 于Thread ，也不 义wait() ， 做 个 ， 个 全可以 ，你 个 候，到 哪个 ？ 了， 不 不 ，只 加

31. wait(), notify() notifyAll()

- 个 wait() 候， 个 ， 会 个 入 到其他 个 上 notify() 同 ， 个 notify() ， 会 个 ，以便其他在 可以 到 个 于 些 ， 只 同 ， 以他们只 在 同 同 块中

32. Thread yield

- 使 前 从 （ ）变为可 （ ）
- 前 到了 ， 么 下 哪个 会从 变 ？可 前 ，也可 其他 ， 分 了

33. Thread sleep() yield ()

- Thread sleep()和 yield() 在 前 在 上 以在其他 于 上 些 义 为什么 些 们可以在 前 在 中 作， 免 员 为可以在其他 些

34. sleep() yield()

- (1) sleep() 其他 会 不 优先 ，因 会 低优先 以 会；yield() 只会 同优先 优先 以 会;
- (2) sleep() 后 入 (blocked) ， yield() 后 入 (ready) ;
- (3) sleep() 出 InterruptedException， yield() 任何 ;
- (4) sleep() yield() (作 CPU 关) 具 可 ， 不 使 yield() 制 发

35.

- 在java中 以下3 可以 在 :
  - 使 出 ，使 出，也 run 后
  - 使 stop ，但 不 个 ，因为stop和suspend及resume 作
  - 使 interrupt 中

36. Java interrupted isInterrupted

- interrupt: 于中 为 为”中 ”

  : 中 仅仅 中 位，不会停 去 为 做 中 (也 中 后会 出interruptedException) 在中 ， 中 为“中 ”， 会 出中

- interrupted: ， 前中 信号 true false 且 中 信号 个 中 了， interrupted 则 回 true， 二 和后 回 false 了
- isInterrupted: 可以 回 前中 信号 true false，与interrupt 别

37.

- 会 不做其他事 ，ServerSocket accept() 回之前， 前 会 ， 到 到 之后 会 回 ， 和 在任务 前 回

38. Java

- 先 ，wait() notify() ， 任 wait() ， 同 也 ， 地， 任 notify() 则

，但 取 ， 到 取 功 下 ；
- 其 ，wait notify 在 synchronized 块 中 ， 且 保 同 块 与 wait notify 同 个， 在 wait 之前 前 功 取 ， wait 后 前 之前 取

39. notify() notifyAll()

- 了 wait() ， 么 便会 于 中， 中 不会去 争
- notifyAll() 会唤 ，notify() 只会唤 个
- notifyAll() 后，会 全 到 ， 后参与 争， 争 功则 ， 不 功则 在 后再 参与 争 notify()只会唤 个 ，具体唤 哪个 制

40.

- 在两个 共享变 即可 共享

一般来说，共享变量要求变量本身是线程安全的，然后在线程内使用的时候，如果有对共享变量的复合操作，那么也得保证复合操作的线程安全性。

41. Java

- 可以 中 和 共享变 和协作
- 典 产 - : 列 ， 产 列 入商品， 在 内， 产 临 (即 列) 占 因为 产 不 临 占 ， 么 列中 商品， 不会 列 ， 么 产 会 下去 因 ， 况下， 列 ，会 产 交出 临 占 ， 入 后 了商品， 后 产 列 了 同 地， 列 ， 也 ， 产 列中 商品了 互 信 协作
- Java中 信协作 :
  - .syncrhoized加 Object wait()/notify()/notifyAll()
    - 二.ReentrantLock 加 Condition await()/signal()/signalAll()
- 交 :
  - 三. 信:

42.

- 同 块 ，因为 不会 住 个 (你也可以 住 个) 同 会 住 个 ，哪 个 中 个不 关 同 块， 会 他们停 个 上
- 同 块 合 原则，只在 住 代 块 住 ， 从侧 也可以 免

请知道一条原则：同步的范围越小越好。

43.

- 个 共享 作 ， 使之 为 个"原 作"，即在 关 作之前，不允 其他 ，否则， 会 坏 ， 会 到 ， 同
- 在 中， 不同 之 同 和 两个 个 之 同 候 会 之 为了 发 ， 同 全
- 互 于共享 ，在各单个 个 使 共享 ，任何 刻 只允 个 去使 ，其 使 ， 到 占 互 可以 同
- 同 体可分为两 ： 和内 名 义，内 利 内 单 同 ，使 切 内 与 ， 不 切 到内 ，只在 作
- 下 ：原 作（例 个单 全 变 ），临 区 内 下 ：事件，信号 ，互
- 同
  - 同 代 ：sychronized 关 修
  - 同 代 块：sychronized 关 修 代 块
  - 使 变 volatile 同 ：volatile关 为 变 供了 免 制
  - 使 入 同 ：reentrantlock 可冲入 互 了lock 口 他与 sychronized 具 同 为和 义

44. (Monitor)

- 在 java 中， 器和 在Java 中 块使 器 块同 代 块， 保 只 个 同 代 块 个 器 和 个 关 在 取 之前不允 同 代
- 代 块 synchronized 修 ， 么 个 分 入了 器 区 ， 保 只 个 分 代 ， 在 取 之前不允 分 代
- 另 java 供了 器(Lock)和 器(synchronized)两

45.

- 可 ：

(1) 使 列 LinkedBlockingQueue，也 列 ， 关 ， 加任务到 列中 ，因为 LinkedBlockingQueue 可以 乎 为 个 列，可以 任务

(2) 使 列 ArrayBlockingQueue，任务 先会 加到ArrayBlockingQueue 中，ArrayBlockingQueue 了，会 maximumPoolSize 值 加 ， 加了 不 ，ArrayBlockingQueue ， 么则会使 RejectedExecutionHandler 了 任务， AbortPolicy

46. servlet ?

- 全 中 , 个 在 中 , 地 个 之 共享变 , 使 功
- Servlet 不 全 , servlet 单 例 , 个 同 同 个 , 不 保 共享变 全
- Struts2 action 例 , 全 , 个 会 new 个 action 分 个 , 后
- SpringMVC Controller 全 吗? 不 , 和 Servlet 似
- Struts2 不 全 ; Servlet 和 SpringMVC 全 , 但 可以 升不 gc，可以使 ThreadLocal

47. Java

- : 使 全 , java.util.concurrent 下 , 使 原 AtomicInteger
- 二: 使 动 synchronized
- 三: 使 动 Lock
- 动 Java 例代 下:

```
  Lock lock = new ReentrantLock();
  lock. lock();
  try {
      System. out. println("获得锁");
  } catch (Exception e) {
      // TODO: handle exception
} finally {
      System. out. println("释放锁");
      lock. unlock();
  }
```

48.

- 个 优先 , , 优先 在 会具 优先 , 但 依 于 , 个 和 作 关 (OS dependent) 们可以 义 优先 , 但 不 保 优先 会在低优先 前 优先 个 int 变 (从 1-10), 1 代 低优先 , 10 代 优先
- Java 优先 会 作 去 , 以与具体 作 优先 关, 别 , 优先
- , 你 优先 可以 setPriority() , 但 了不 会 变, 个 不准

49.

- 个 和 住: 块 new 个 在 , run 代
- 上 你 到困 , 么 举个例 , 假 Thread2 中 new 了Thread1，main 函中 new 了 Thread2, 么:

(1) Thread2 块 main , Thread2 run() Thread2

（2）Thread1 块 Thread2 ，Thread1 run() Thread1

50. Java dump Java

- Dump 件 内 像 可以 器保 到dump 件中
- 在 Linux 下，你可以 命令 kill -3 PID （Java ID） 取 Java dump 件
- 在 Windows 下，你可以 下 Ctrl + Break 取 JVM 会 dump 件 印到准 出 件中， 可 印在 制台 件中，具体位 依

51.

- 会停 Thread.UncaughtExceptionHandler 于中 况 个内 口 个 中 候，JVM 会使 Thread.getUncaughtExceptionHandler() UncaughtExceptionHandler 和 作为参 传 handler uncaughtException()

52. Java

- 命周
- CPU

  可 于可 器 ， 么 会 会占内 ， 垃圾回 器 压力， 且 在 争 CPU 产 其他

- 低 JVM

  在可创 上 在 个 制， 个 制值 台 不同 不同， 且 受 个因制 ，包 JVM 启动参 Thread 函 中 ，以及 作 制坏了 些 制， 么可 出OutOfMemoryError

53.

sleep()	个 N
isAlive()	判 个 否
join()	
activeCount()	中
enumerate()	举 中
currentThread()	到 前
isDaemon()	个 否为
setDaemon()	个 为
setName()	为 个名
wait()	个
notify()	个
setPriority()	个 优先

---

1. Java

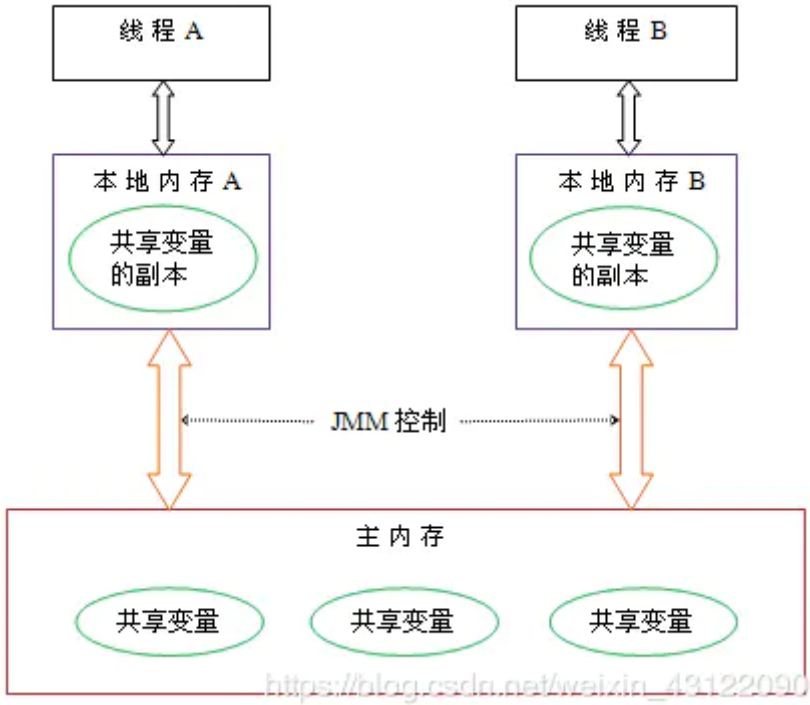
- 垃圾回 在内 中 在 作
- 垃圾回 别 且丢 不再使 和

2.

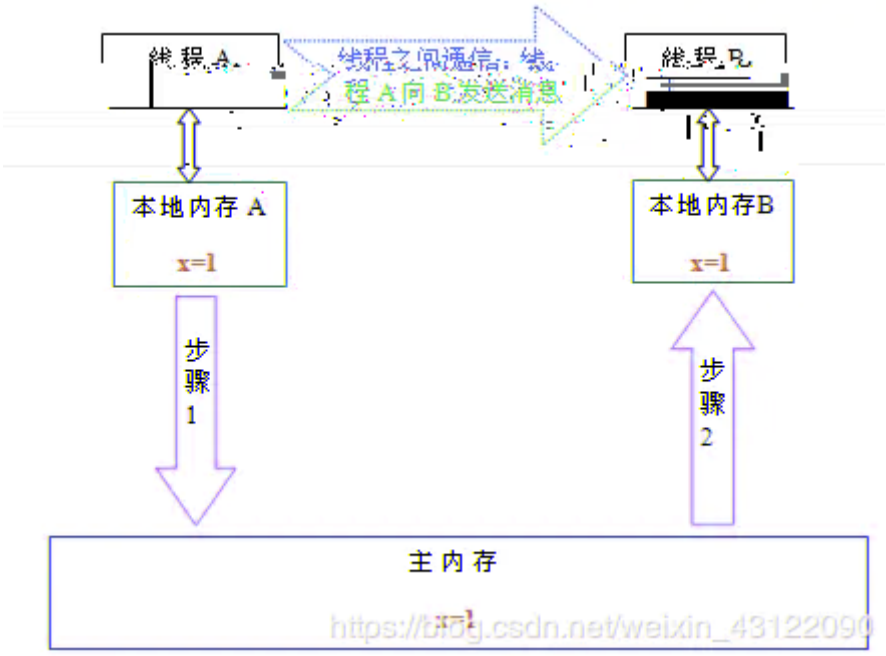
- 在 发 中， 们 两个关 ： 之 何 信及 之 何同 信 之 以 何 交 信 之 信 制 两 ：共享内 和 传
- Java 发 共享内 ，Java 之 信 ， 个 信 员 全 写 Java 员不 之 信 作 制， 可 会 到各 内 可

3. Java

- 共享内 Java内 （ JMM)，JMM决 个 共享变 写入 ， 另 个 可 从 ，JMM 义了 和主内 之 关 ： 之 共享变 储在主内 （main memory）中， 个 个 地内 （local memory），地内 中 储了 以 /写共享变 副 地内 JMM 个 ， 不 在 了 ，写 冲区， 器以及其他 件和 器优化



- 从上图 ， A与 B之 信 ， 历下 2个 ：
  1. 先， A 地内 A中 共享变 刷 到主内 中去
  2. 后， B到主内 中去 取 A之前 共享变



- ：什么 Java内 ：java内 jmm， 义了 个 另 个 可 共享变 在主内 中， 个 地内 ， 个 同 个 候，可地内 及 刷 到主内 ， 以 会发 全

4. null

- 不会，在下 个垃圾回 周 中， 个 可回
- 也 不会 即 垃圾 器 刻回 ， 在下 垃圾回 会 其占 内

5. finalize() (finalization)

- 1.垃圾回 器（garbage colector）决 回 , 会 finalize() ; finalize Object 个 , 在Object 中 protected void finalize() throws Throwable { } 在垃圾回 器 会 回 finalize() , 可以 其 回 : 垃圾回 器准 占 内 , 先 finalize() , 且下 垃圾回 动作发 , 回 占 内
- 1. GC 内 回 了, 在finalization做什么 ? 分 候，什么 不 做(也 不) 只 在 些 况下, 你 了 些native (C写)，可以 在finaliztion 去 C 函
  - Finalizetion主 占 (不 内 , 其他 , 件(File Handle) 口(ports) (DB Connection)) , 不 地 作

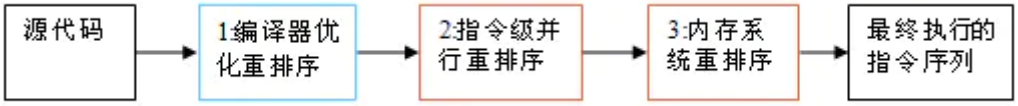
6.

- 代 先后
- 器为了 ，可 会 入代 优化, ( )， 不保 中各个 句 先后 同代 中 ，但 会保 和 代

```
int a = 5;    //语句1
int r = 3;    //语句2
a = a + 2;    //语句3
r = a*a;      //语句4
```

- 则因为 ，他 可 为（ 句 ） 2-1-3-4，1-3-2-4 但 不可 2-1-4-3，因为 了依 关
- 单 不会 任何 ，但 不 了， 以 们在 个 了

7.



1. 器优化 器在不 变单 义 前 下，可以 句
2. 令 代 器 了 令 (ILP) 令 叠 不 在 依 , 器可以 变 句 器 令
3. 内 于 器使 和 /写 冲区, 使 加 和 储 作 上去可 在 乱

- 些 于单 ，但 可 会 出 内 可

8.

- as-if-serial：

1. 不 么 , 不 变
2. 不 在 依 可以 器和 器
3. 个 作依 两个 作, 两个 作 不 在依 可以
4. 单 则不会 , 但 后 会

9. as-if-serial happens-before

- as-if-serial 义保 单 内 不 变，happens-before关 保 同 不 变
- as-if-serial 义 写单 员创 了 个 ：单 happens-before关 写 同 员创 了 个 ： 同 happens-before
- as-if-serial 义和happens-before 么做 , 为了在不 变 前 下，可 地

10. synchronized

- 在 Java 中，synchronized 关 制 同 , 在 下, 制 synchronized 代 不 个 同 synchronized 可以修 变
- 另 ，在 Java 中，synchronized 于 , 低下，因为 器 (monitor) 依 于 作 Mutex Lock ，Java 到 作 原 之上 唤 个 , 作 , 作 之 切 从 到内 , 个 之 , , 也 为什么 synchronized 低 原因 在 Java 6 之后 Java 从 JVM synchronized 优化, 以 在 synchronized 也优化 不 了 JDK1.6 入了 优化, 化 偏向 减 作

11. synchronized

synchronized

- 修 例 : 作 于 前 例加 , 入同 代 前 前 例
- 修 : 也 前 加 ，会作 于 例，因为 员不 于任何 个 例 , 员（static 个 ，不 new了 个 ，只 份） 以 个 A 个 例 synchronized , B 个 例 synchronized , 允 ，不会发 互 ，因为 synchronized 占 前 , synchronized 占 前 例
- 修 代 块: 加 , 加 , 入同 代 前

`总结： synchronized 关键字加到 static 静态方法和 synchronized(class)代码块上都是是给 Class 类上锁。synchronized 关键字加到实例方法上是给对象实例上锁。尽量不要使用 synchronized(String a) 因为JVM中，字符串常量池具有缓存功能！`

12. ”

- 双 制 出 为了 决前 同 和 , 下 代 , 单分 下 决了 不会出 new , 且也 了 加

```
  public class Singleton {
     private volatile static Singleton uniqueInstance;
     private Singleton() {}

    public static Singleton getUniqueInstance() {
            //先判断对象是否已经实例过，没有实例化过才进入加锁代码
if (uniqueInstance == null) {
             //类对象加锁
synchronized (Singleton.class) {
                 if (uniqueInstance == null) {
                     uniqueInstance = new Singleton();
                 }
             }
         }
        return uniqueInstance;
    }
```

}

另外，需要注意 uniqueInstance 采用 volatile 关键字修饰也是很有必要。

- uniqueInstance volatile 关 修 也 , uniqueInstance = new Singleton(); 代 其 分为三 :

1. 为 uniqueInstance 分 内
2. 初 化 uniqueInstance
3. uniqueInstance 向分 内 地址

但是由于 JVM 具有指令重排的特性，执行顺序有可能变成 1->3->2。指令重排在单线程环境下不会出现问题，但是在多线程环境下会导致一个线程获得还没有初始化的实例。例如，线程 T1 执行了 1 和 3，此时 T2 调用 getUniqueInstance() 后发现 uniqueInstance 不为空，因此返回 uniqueInstance，但此时 uniqueInstance 还未被初始化。

使用 volatile 可以禁止 JVM 的指令重排，保证在多线程环境下也能正常运行。

13. synchronized

- Synchronized 义 个monitor（ 器 ） ，
- 个 个 器 (monitor) 个Synchronized修 代 monitor 占 会 于 且 取monitor , :

  1 monitor 入 为0，则 入monitor， 后 入 为1， 即为 monitor

  2 占 monitor，只 入，则 入monitor 入 加1.

  3 其他 占 了monitor，则 入 , 到monitor 入 为0，再 取monitor

synchronized是可以通过 反汇编指令 javap命令，查看相应的字节码文件。

14. synchronized

- 入 个 取到 之后， 可以 原 个 器，
取 ， 器加 ，再 加 ， ， 器减 ， 器值为0
， 任何 ，其 可以 争 取

15.

- synchronized 代 只 些 单 代 ， ， 加
可 不 值 作，因为 及到 和内 切
synchronized 代 ，不 不 ， 在 synchronized
做 ， 做了 发 ，再 ， 可

- ： 员 个 ，不像传 wait(), sleep() yield() 们 了
CPU 制， 不会 CPU， 在 个 么做 为了保 CPU
，在 中， 个 候可 会在另 个内 ， 会 为了
免 和减 可以使 了

16. synchronized

- synchronized 升 原 ：在 个 threadid ，在 候
threadid 为 ，jvm 其 偏向 ， threadid 为其 id，再 入 候会先判
threadid 否与其 id ， 则可以 使 ， 不 ，则升 偏向 为
， 取 ， 之后， 取到 使
， 会 从 升 为 ， 了 synchronized 升

锁的升级的目的：锁升级是为了减低了锁带来的性能消耗。在 Java 6 之后优化 synchronized 的实现方式，使用了偏向锁升级为轻量级锁再升级到重量级锁的方式，从而减低了锁带来的性能消耗。

- 偏向 ， 名 义， 会偏向于 个 ， 在 中，同 只 个
，不 在 争 况，则 不 发同 ，减 加 / 些CAS 作（
列 些CAS 作）， 况下， 会 加 个偏向 在 中，
到了其他 占 ，则 偏向 会 ，JVM会 上 偏向 ， 到
准
- 偏向 升 ，偏向 在 个 入同 块 况下， 二个 加入
争 候， 会升 为 ；
- synchronized ， Java 中 为 在 下，Java 会
加 ， 且在 候，唤 些

17. B A

- (1) volatile 修 变
- (2) synchronized 修 修 变
- (3) wait/notify
- (4) while

18. synchronized A synchronized B

- 不 其 只 同 ，同 则不 入 因为 上 synchronized 修 ， 入A 取 ， 么 图 入B 只 在 （ 不 哦）中

19. synchronized volatile CAS

- （1）synchronized ， 于 占 ，会 其他
- （2）volatile 供 共享变 可 和 令 优化
- （3）CAS 于冲 乐 （ ）

20. synchronized Lock

- 先synchronized Java内 关 ，在JVM ，Lock 个Java ；
- synchronized 可以 代 块加 ； lock 只 代 块加
- synchronized 不 动 取 和 ，使 单，发 会 动 ，不会 ； lock 加 和 ， 使 不 unLock()去 会
- Lock 可以 功 取 ， synchronized 却 办到

21. synchronized ReentrantLock

- synchronized 和 if else for while 关 ，ReentrantLock ， 二 区别 ReentrantLock ， 么 供了 synchronized ，可以 可以 可以 各 各 变
- synchronized 低 ， ReentrantLock， 场 ，但 在 Java 6 中 synchronized 了
- 同 ：两 可 入

两 可 入 “可 入 ” ： 可以再 取 内 个 了 个 ， 个 ， 其再 取 个 候 可以 取 ， 不可 入 ， 会 同 个 取 ， 器 1， 以 到 器 下 为0

- 主 区别 下:
  - ReentrantLock 使 ，但 合动作;
  - ReentrantLock 动 取与 ， synchronized 不 动 和 启 ；
  - ReentrantLock 只 于代 块 ， synchronized 可以修 变
  - 二 制其 也 不 ReentrantLock Unsafe park 加 ， synchronized 作 中 mark word
- Java中 个 可以作为 ， synchronized 同 ：
  - 同 ， 前 例
  - 同 ， 前 class
  - 同 块， 号

22. volatile

- 于可 ，Java 供了 volatile 关 保 可 和 令 volatile 供 happens-before 保 ， 保 个 修 其他 可 个共享变 volatile 修 ， 会保 修 值会 即 到主内 中， 其他 取 ， 会去内 中 取值
- 从 ，volatile 个 作 和 CAS 合，保 了原 ， 可以参 java.util.concurrent.atomic 包下 ， AtomicInteger
- volatile 于 下 单 作(单 单 写)

23. Java volatile

- ，Java 中可以创 volatile ，不 只 个 向 ， 不 个 ， 变 向 ， 会受到 volatile 保 ，但 个 同 变 元 ，volatile 不 到之前 保 作 了

24. volatile atomic

- volatile 变 可以 保先 关 ，即写 作会发 在后 作之前, 但 不 保 原 例 volatile 修 count 变 ， 么 count++ 作 不 原
- AtomicInteger 供 atomic 可以 作具 原 getAndIncrement() 会原 作 前值加 ，其 和 变 也可以 似 作

25. volatile

- 关 volatile 主 作 使变 在 个 可 ，但 保 原 ， 于 个 同 个 例变 加 同
- volatile只 保 可 不 保 原 ，但 volatile修 long和double可以保 其 作原

Oracle Java Spec

- 于64位 long和double， volatile修 ， 么 其 作可以不 原 在 作 候，可以分 两 ， 32位 作
- 使 volatile修 long和double， 么其 写 原 作
- 于64位 地址 写， 原 作
- 在 JVM ，可以 否 写long和double作为原 作
- JVM 为原 作

26. synchronized volatile

- synchronized 只 个 可以 取作 ， 代 ， 其他
- volatile 变 在 CPU 器中 不 ， 从主 中 取 保 下变 可 ； 令

- volatile 变 修 ；synchronized 可以修 变
- volatile 仅 变 修 可 ，不 保 原 ； synchronized 则可以保 变 修 可 和原
- volatile 不会 ；synchronized 可 会
- volatile 变 不会 器优化；synchronized 变 可以 器优化

- volatile关 同 , 以volatile synchronized关 但 volatile关 只 于变 synchronized关 可以修 以及代 块 synchronized关 在JavaSE1.6之后 了主 包 为了减 和 入 偏向 和 以及其 各 优化之后 了 升, 发中使 synchronized 关 场 些

27. final

- 不可变 (Immutable Objects)即 创 (, 也即 值) 不 变，反之即为可变 (Mutable Objects)
- 不可变 即为不可变 (Immutable Class) Java 台 中包含 不可变 , String 包 BigInteger 和 BigDecimal
- 只 下 , 个 不可变 ;
  - 不 在创 后再 修 ;
  - final ; 且, 创 (创 发 this 出)

不可变对象保证了对象的内存可见性，对不可变对象的读取不需要进行额外的同步手段，提升了代码执行效率。

28. Lock synchronized

- Lock 口 同 和同 块 供了 具 作 他们允 ，可以具 全不同 , 且可以 个 关 件
- 优势 :
  - (1) 可以使 公
  - (2) 可以使 在 候响 中
  - (3) 可以 取 , 在 取 候 即 回
  - (4) 可以在不同 围，以不同 取和
- 体上 Lock synchronized , Lock 供了 件 可 (tryLock) (tryLock 参) 可中 (lockInterruptibly) 可 件 列 (newCondition) 作 另 Lock 公 ( )和公 ，synchronized 只 公 , ，在 分 况下, 公

29.

- : 假 坏 况, 去 候 为别人会修 , 以 在 候 会上 , 别人 个 会 到 到 传 关 到了 制, , , ，写 , 在做 作之前先上 再 Java 同 原 synchronized 关 也
- 乐 : 名 义, 乐 , 去 候 为别人不会修 , 以不会上 ，但 在 候会判 下在 别人 去 个 ，可以使 号 制 乐 于 , 可以 吞吐 ，像 供 似于 write_condition 制，其 供 乐 在 Java中 java.util.concurrent.atomic 包下 原 变 使 了乐 CAS

30. CAS

- CAS compare and swap 写，即 们 交

- Atomic包中 在 下， 个 同 单个（包 及 ）变 作 ，具 他 ，即 个 同 变 值 ，仅 个 功， 功 可以向 ， ， 到 功

35.

- ： 两个 两个以上 （ ）在 中，因争 互 ， 力作 ， 们 下去
- ：任务 ， 于 些 件 ， ， ， ，
- 和 区别在于， 于 体 在不 变 ， “ ”， 于 体 为 ； 可 ， 则不
- ： 个 个 因为 原因 ，

  Java 中 原因：
  - 1 优先 吞 低优先 CPU
  - 2 久 在 个 入同 块 ，因为其他 在 之前 地 同 块
  - 3 在 个 也 于 久 （ 个 wait ），因为其他 地 唤

1.

- Java中 场 发 ，几乎 发 任务 可以使 在 发 中，合 地使
  - 低 利 创 低 创 和
  - 响 任务到 ，任务可以不 到 创 即
  - 可 ， 制地创 ，不仅会 ， 会低 ，使 可以 分 优和 但 ， 做到合 利

2.

- 为 发 ， 几个固 为 作 务，减 了创和 ，从
- 个 ， 了(不 不 作 作，不 低 创 和 ， 你 么久 不 制 不 创)，况且 们不 制 中 和中

3.

- 低 ： 在 ，减 创
- 响 可 制 发 ， 使 ，同 免 争，免 任务到 ，任务可以不 到 创 即
- 可 ， 制 创 ，不仅会 ， 会 低 ，使 可以 分 ， 优和

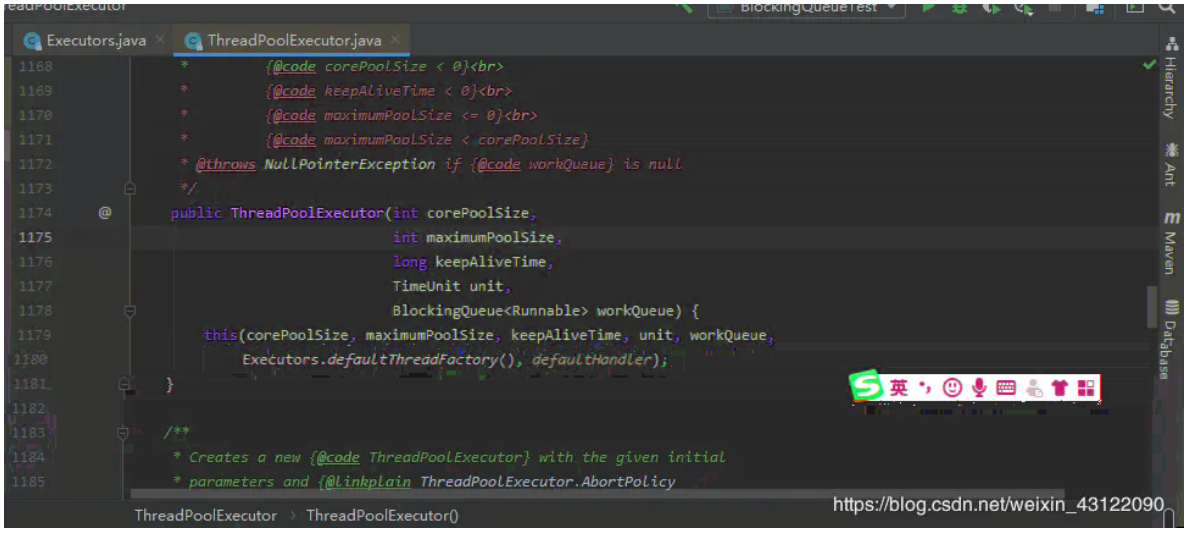
- 加功 ： 供 单 发 制 功

4. ThreadPoolExecutor

- **ThreadPoolExecutor**

  ThreadPoolExecutor其 也 JAVA 个 ， 们 Executors 厂 ， 传入 不同 参 ， 可以 出 于不同 场 下 ThreadPoolExecutor （ ）

  参 图:



构造参数参数介绍:

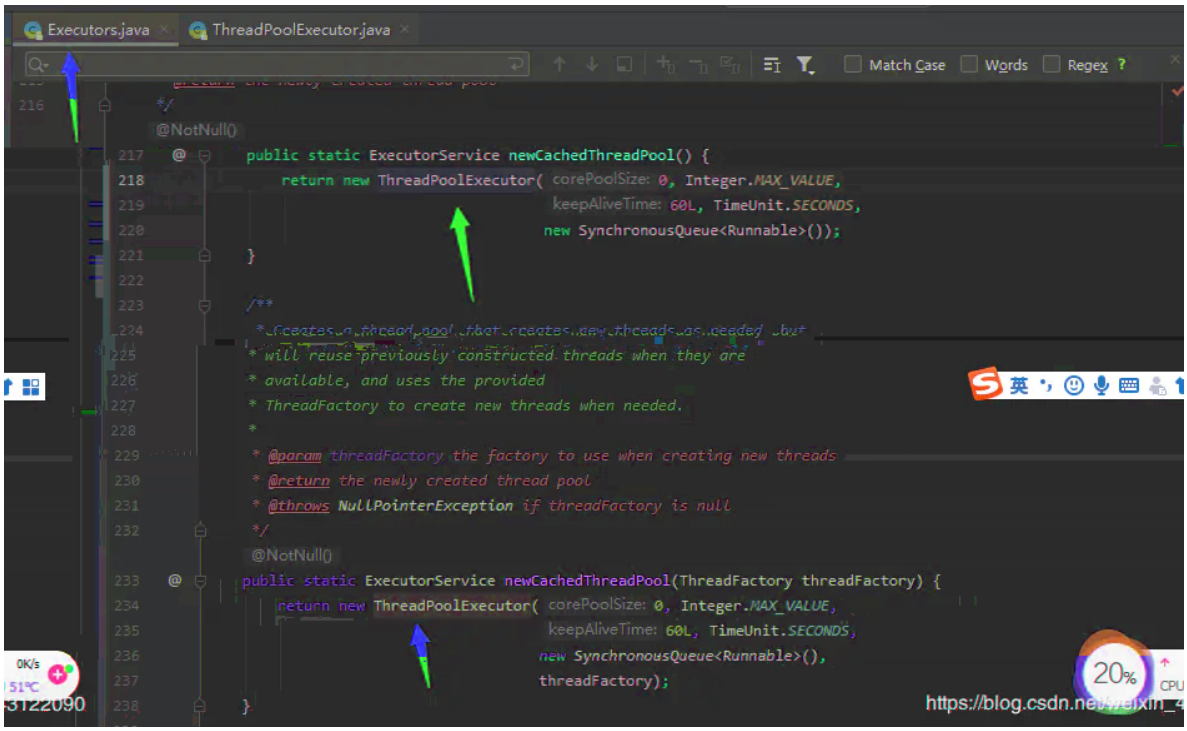
```
corePoolSize 核心线程数量
maximumPoolSize 最大线程数量
keepAliveTime 线程保持时间，N个时间单位
unit 时间单位（比如秒，分）
workQueue 阻塞队列
threadFactory 线程工厂
handler 线程池拒绝策略
```

5. Executors

- **Executors**

  Executors 厂 中 供 newCachedThreadPool newFixedThreadPool newScheduledThreadPool newSingleThreadExecutor 其 也只 ThreadPoolExecutor 函 参 不同 传入不同 参 ， 可以 出 于不同 场 下 ，

Executor 厂 何创 图:



6.

- **Java Executors jdk1.5**
  1. newCachedThreadPool创 个可 ， ，可 回 ， 可回 ，则
  2. newFixedThreadPool 创 个 ，可 制 发 ， 出 会在 列中
  3. newScheduledThreadPool 创 个 ， 及周 任务
  4. newSingleThreadExecutor 创 个单 化 ， 只会 唯 作 任务，保 任务 (FIFO, LIFO, 优先)

7. Java Executor Executors

- Executors 具 不同 们 创 了不同 ， 业务
- Executor 口 们 任务
- ExecutorService 口 了 Executor 口 了 ， 供了 们 任务 且可以 取任务 回值
- 使 ThreadPoolExecutor 可以创 义

8.

1. newCachedThreadPool

- ：newCachedThreadPool创 个可 ， 前 了 ， 可以 回 ， 加 ， 可以 加 ， 不会 作任何 制
- ：他 可以 ，但 内 出，因为 值 在初 化 候 为 Integer.MAX_VALUE， 器 么 内 不 使 可 出 ， 可以去 写 个 制 下 个 值

- : 为 , 二个任务 个任务 , 会 个任务 , 不
-

```
package com.lijie;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TestNewCachedThreadPool {
    public static void main(String[] args) {
        // 创建无限大小线程池，由jvm自动回收
ExecutorService newCachedThreadPool = Executors.newCachedThreadPool();
          for (int i = 0; i < 10; i++) {
              final int temp = i;
              newCachedThreadPool.execute(new Runnable() {
                  public void run() {
                      try {
                          Thread.sleep(100);
                      } catch (Exception e) {
                      }
                      System.out.println(Thread.currentThread().getName() +
",i==" + temp);
                  }
              });
          }
      }
  }
```

2.newFixedThreadPool

- : 创 个 , 可 制 发 , 出 会在 列中
- : 固 , 但 列 列 压, 列 , OOM (出内)
- : 压 和分 匹 , Runtime.getRuntime().availableProcessors()

`Runtime.getRuntime().availableProcessors()方法是查看电脑CPU核心数量）`

-

```
package com.lijie;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TestNewFixedThreadPool {
    public static void main(String[] args) {
        ExecutorService newFixedThreadPool =
Executors.newFixedThreadPool(3);
        for (int i = 0; i < 10; i++) {
            final int temp = i;
            newFixedThreadPool.execute(new Runnable() {
                public void run() {
```

```
                    System.out.println(Thread.currentThread().getName() +
",i==" + temp);
                }
            });
        }
    }
}
```

3.newScheduledThreadPool

- ：创 个固 ， 且 以及周 任务 ， 似于 Timer（Timer Java 个 器 ）
- ： 于 任务 同 个 ，因 任务 串 ，同 只 个任务在 ，前 个任务 会 响到之后 任务（ ： 个任务出 ，以后 任务 ）
-

```
package com.lijie;

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class TestNewScheduledThreadPool {
    public static void main(String[] args) {
        //定义线程池大小为3
        ScheduledExecutorService newScheduledThreadPool =
Executors.newScheduledThreadPool(3);
        for (int i = 0; i < 10; i++) {
            final int temp = i;
            newScheduledThreadPool.schedule(new Runnable() {
                public void run() {
                    System.out.println("i:" + temp);
                }
            }, 3, TimeUnit.SECONDS);//这里表示延迟3秒执行。
}
      }
  }
```

4.newSingleThreadExecutor

- ：创 个单 化 ， 只会 唯 作 任务， 个唯 因为 ， 么会 个 代 ，他 保 前 任务 后 保 任务 (FIFO, LIFO, 优先)
- ： ， ，他 单 ， 发业务下 力
- ：保 任务 ， 个唯 因为 ， 么会 个 代
-

```
package com.lijie;

import java.util.concurrent.ExecutorService;
```

```
import java.util.concurrent.Executors;

public class TestNewSingleThreadExecutor {
    public static void main(String[] args) {
        ExecutorService newSingleThreadExecutor =
Executors.newSingleThreadExecutor();
        for (int i = 0; i < 10; i++) {
            final int index = i;
            newSingleThreadExecutor.execute(new Runnable() {
                public void run() {
                    System.out.println(Thread.currentThread().getName() + "
index:" + index);
                    try {
                        Thread.sleep(200);
                    } catch (Exception e) {
                    }
                }
            });
        }
    }
}
```

9.

- RUNNING: , 受 任务, 列中 任务
- SHUTDOWN: 不 受 任务 交, 但 会 列中 任务
- STOP: 不 受 任务 交, 不再 列中 任务, 中 在 任务
- TIDYING: 任务 了, workCount 为 0, 在 为 TIDYING , 会 terminated()
- TERMINATED: terminated() 后, 会变 个

10. submit() execute()

- 同 :
  - 同 可以 启 中 任务
- 不同 :
  - 参 : execute()只 Runnable 任务 submit()可以 Runnable 和 Callable 任务
  - 回值: submit() 可以 回 Future , execute()
  - : submit() 便Exception

11. Java

- ThreadGroup , 可以 到 个 中, 中可以 , 也可以 , 中 可以 , 似于
- 和 两个不同 , 他们 作 全不同, 前 为了 便 , 后 为了 命周 , , 减 创
- 为什么不 使 ? 因为使 全 吧, 具体 , 使 , 使

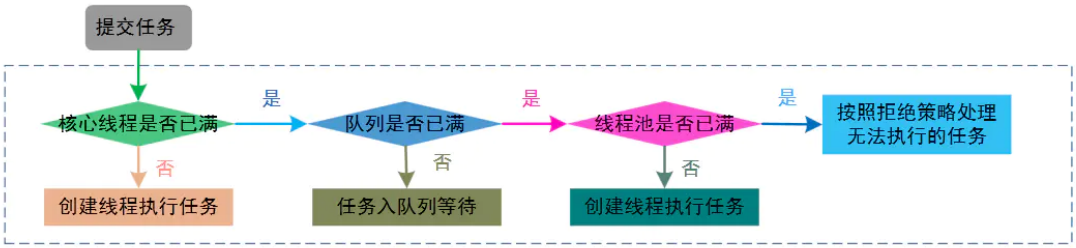
12. ThreadPoolExecutor

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(1, 2, 60L,
TimeUnit.SECONDS, new ArrayBlockingQueue<>(3));
          for (int i = 1; i <= 6; i++) {
              TaskThred t1 = new TaskThred("任务" + i);
              //executor.execute(t1);是执行线程方法
executor.execute(t1);
          }
          //executor.shutdown()不再接受新的任务，并且等待之前提交的任务都执行完再关
闭，阻塞队列中的任务不会再执行。
executor.shutdown();
      }
  }

  class TaskThred implements Runnable {
      private String taskName;

      public TaskThred(String taskName) {
          this.taskName = taskName;
      }
      public void run() {
          System.out.println(Thread.currentThread().getName() + taskName);
      }
  }
```

14.



- 交 个任务到 中， 下：

  1. 判 否 在 任务， 不 （
     创 ）则创 个 作 任务 在 任务，则 入下个

  2. 判 作 列 否 ， 作 列 ，则 交 任务 储在 个 作
     列 作 列 了，则 入下个
  3. 判 否 于 作 ， ，则创 个 作 任
     务 了，则交 和 个任务

15. ?

- 合 分 况 ， 单 CPU 和IO
  分

  CPU

- CPU 任务 ， ，CPU 全

- CPU 任务只 在 CPU上 可 到加 ( ), 在单 CPU上, 你 几个 , 任务 不可 到加 , 因为CPU 力

IO

- IO , 即 任务 IO, 即 在单 上 IO 任务会 CPU 力 在 以在IO 任务中使 可以 加 , 即 在单 CPU上, 加 主 利 了

CPU IO

1. CPU , 任务可以 , 和 器 cpu , 可以使 个 在 任务
2. IO , 分 , , 2*cpu

- 从以下几个 分 任务 :
  - 任务 : CPU 任务 IO 任务 合 任务
  - 任务 优先 : 中 低
  - 任务 : 中
  - 任务 依 : 否依 其他 ,

- CPU 例 ,
- CPU 例 ,

>

1.

- : Vector ConcurrentHashMap HasTable

- 件 发中 器 HashMap ArrayList，LinkedList，
- 但 在 发中 不 乱 器， 使 了 加 （ 同 ） 合，你 会
  乱 在 发中 使 器 加 （同 ） 器

2. Vector

- Vector与ArrayList ，也 ，不同 同 ，即 刻只
  个 写Vector， 免 同 写 不 ，但 同 ，
  ArrayList

（ArrayList是最常用的List实现类，内部是通过数组实现的，它允许对元素进行快速随机访问。当从ArrayList的中间位置插入或者删除元素时，需要对数组进行复制、移动、代价比较高。因此，它适合随机查找和遍历，不适合插入和删除。ArrayList的缺点是每个元素之间不能有间隔。）

3. ArrayList Vector

- Vector 上了synchronized关 ， 同

1. ArrayList 加

```
public boolean add(E e) {
    modCount++;
    add(e, elementData, size);
    return true;
}
```

2. Vector 加 （加 了synchronized关 ）

```
public synchronized boolean add(E e) {
    modCount++;
    add(e, elementData, elementCount);
    return true;
}
```

4. HashTable

- 因为HasTable 内 synchronized修 了， 以 全 其他 和HashMap

1. HashMap 加

```
public V put(K key, V value) { return putVal(hash(key), key, value, onlyIfAbsent: false, evict: true); }

/**
 * Implements Map.put and related methods.
```

2. HashTable 加

```
public synchronized V put(K key, V value) {
    // Make sure the value is not null
    if (value == null) {
        throw new NullPointerException();
    }
```

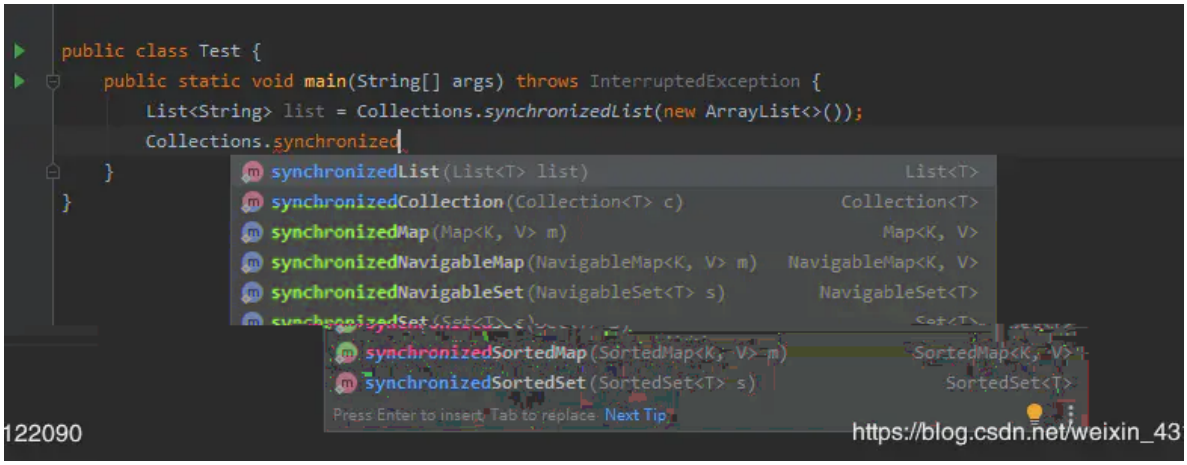
5. ConcurrentHashMap HashTable

- ConcurrentHashMap Java5中 发 吞吐 全HashMap Segment 和HashEntry Segment 在ConcurrentHashMap ， HashEntry则 于 储 -值 个ConcurrentHashMap 包含 个Segment ， Segment 和HashMap 似， 和 ； 个Segment 包含 个HashEntry ， 个HashEntry 个 元 ； 个Segment 个HashEntry 元 ， HashEntry 修 ， 先 Segment
- 不 ？？？ ， 也 不
- ：
  1. HashTable 了HashMap加上了synchronized， ConcurrentHashMap 分 + ， 全
  2. ConcurrentHashMap 个Map分为N个Segment，可以 供 同 全，但 升N倍， 升16倍
  3. 且 作不加 ， 于HashEntry value变 volatile ，也 保 取到 值
  4. Hashtable synchronized Hash ，即 住 占， ConcurrentHashMap允 个修 作 发 ，其关 在于使 了 分
  5. ： 内 （ 内元 Entry 75% 发 ，不会 个Map ），入前 不 ， 免

6. Collections.synchronized *

注意：* 号代表后面是还有内容的

- 什么 ，他 全全 可以 List Map Set 口 下 合变 全 合
- Collections.synchronized *： 原 什么， 代



7. Java ConcurrentHashMap

- ConcurrentHashMap map 划分 分 可 和 全 划分 使 发 , ConcurrentHashMap 函 个可 参 , 值为 16, 在 况下 免争
- 在 JDK8 后, 了 Segment ( ) , 启 了 全 ,利 CAS 同 加入了 助变 发 , 具体内 吧

8.

- 何为同 器: 可以 单地 为 synchronized 同 器, 个 同 器 , 们 会串 Vector, Hashtable, 以及 Collections.synchronizedSet, synchronizedList 回 器 可以 Vector, Hashtable 些同 器 代 , 可以 到 些 器 全 们 , 在 同 上加上关 synchronized
- 发 器使 了与同 器 全不同 加 供 发 和伸 , 例 在 ConcurrentHashMap 中 了 加 制, 可以 为分 , 在 制下, 允 任 发地 map, 且 作 和写 作 也可以 发 map, 同 允 写 作 发地修 map, 以 可以在 发 下 吞吐

9. Java

- 同 合与 发 合 为 和 发 供了合 全 合, 不 发 合 可 在 Java1.5 之前 员们只 同 合 且在 发 候会 争 , 了 Java5 介 了 发 合像ConcurrentHashMap, 不仅 供 全 分 和内 分区 代 了可

10. SynchronizedMap ConcurrentHashMap

- SynchronizedMap 住 保 全, 以 只 个 为 map
- ConcurrentHashMap 使 分 保 在 下
- ConcurrentHashMap 中则 住 个 ConcurrentHashMap hash 分为 16 个 , get, put, remove 作只 前 到
- , 原 只 个 入, 在却 同 16 个写 , 发 升

- 另 ConcurrentHashMap 使 了 不同 代 在 代 中, iterator 创 后 合再发 变 不再 出ConcurrentModificationException, 取 代之 在 变 new 从 不 响原 , iterator 后再 为 , iterator 可以使 原 , 写 也可以 发 变

11. CopyOnWriteArrayList ?

- CopyOnWriteArrayList 个 发 器 人 全 , 为 句 不严 , 个前 件, 合场 下 作 全
- CopyOnWriteArrayList(免 器) 之 个 代器同 历和修 个列 , 不会出 ConcurrentModificationException 在CopyOnWriteArrayList 中, 写入 创 个 副 , 保 在原地, 使 制 在 修 , 取 作可以 全地

12. CopyOnWriteArrayList ?

- 合 写 场

13. CopyOnWriteArrayList ?

- 于写 作 候， ，会 内 ， 原 内 况下，可 young gc full gc
- 不 于 场 ，像 元 ， 以 个 set 作后， 取到 可 ， CopyOnWriteArrayList 做到 ,但
- 于 使 中可 保 CopyOnWriteArrayList 到 ，万 ， add/set 制 ， 个代价 在 了 在 互 中， 作分分

14. CopyOnWriteArrayList ?

- 写分 ， 和写分
-
- 使 另 ， 决 发冲

1.

- 列 人 ： 列 分 中 件， 与 信
- 发 列 什么： 发 列 个 以 共享 件

2.

那就有可能要说了，我们并发集合不是也可以实现多线程之间的数据共享吗，其实也是有区别的：

- 列 “先 先出” 则，可以 ， 列 决 和
- 发 合 在 个 中共享

3.

- 在 发 列上JDK 供了Queue 口， 个 以Queue 口下 BlockingQueue 口为代 列，另 个 （ ） 列

4.

- 列 列为 ，从 列中 取元 作 会
- 列 ， 列 加元 作会
- 图从 列中 取元 会 ， 到其他 列 入 元

- 图 列中 加 元 同 也会 , 到其他 使 列 变

5.

1.

   1. **ArrayDeque,**

      ArrayDeque (列) JDK 器中 个双 列 , 内 使 元 储, 不允 储null值, 可以 元 和 入取出, 作 列 双 列 佳 , LinkedList

2. **PriorityQueue,**

   PriorityQueue (列) 个 于优先 优先 列 优先 列 元 其 , 列 供 Comparator , 具体取决于 使 列不允 使 null 元 也不允 入不可

3. **ConcurrentLinkedQueue,**

   ConcurrentLinkedQueue (列): 个 于 发场 下 列, , 了 发 下 ConcurrentLinkedQueue 于BlockingQueue 口, 个 于 全 列 列 元 先 先出 原则 列不允 null元

4.

   1. **DelayQueue,**

      DelayQueue 个 BlockingQueue , 加入其中 元 Delayed 口 产 put之 加入元 , 会 发Delayed 口中 compareTo , 也 列中元 到 , 们 入 列 在 列 元 到 , 后到

5. **ArrayBlockingQueue,**

   ArrayBlockingQueue 个 列, 内 个 , 们 在其初 化 候 , 不可 变 ArrayBlockingQueue 以先 先出 储

6. **LinkedBlockingQueue,** **FIFO**

   LinkedBlockingQueue 列 可 , 们初 化 个 , , 不 , , 其 了 为 Integer.MAX_VALUE 内 个

7. **LinkedBlockingDeque,** **FIFO**

   LinkedBlockingDeque 个 双向 列, 即可以从 列 两 入和 元 双向 列因为 了 个 作 列 入口, 在 同 入 , 也 减 了 半 争

于其他 列, LinkedBlockingDeque 了addFirst addLast peekFirst peekLast , 以first , 入 取 双 列 个元 以last , 入 取 双 列 后 个元

LinkedBlockingDeque 可 , 在初 化 可以 其 , 不 , 为Integer.MAX_VALUE

5. **PriorityBlockingQueue,**

priorityBlockingQueue 个 列， 制，在内 允 况下可以 加元 ；又 具 优先 列， 函 传入 判 ，传入 comparable 口

6. **SynchronousQueue**

SynchronousQueue 个内 只 包含 个元 列 入元 到 列 ， 到另 个 从 列中 取了 列中 储 元 同 ， 取元 且 前不 在任何元 ，则 ， 到 元 入 列

个 为 列 其 像 个

不管是那种列队，是那个类，当是他们使用的方法都是差不多的

add()	在不 出 列 况下 入元 ，可以 即 ， 功 回true， 列 了 出
offer()	在不 出 列 况下 入元 候则可以 即在 列 入 元 ， 功 回true， 列 ，则 回false
put()	入元 候， 列 了 ， 到 列可
take()	从 列中 取值， 列中 值， 会 ， 到 列中 值， 且 取 了 值
poll(long timeout, TimeUnit unit)	在 ，从 列中 取值， 取到会 出
remainingCapacity()	取 列中剩余
remove(Object o)	从 列中 值
contains(Object o)	判 列中 否 值
drainTo(Collection c)	列中值，全 ， 发 到 合中

1.

- CountDownLatch

  CountDownLatch 位于java.util.concurrent包下，利 可以 似 器 功 个任务A， 其他3个任务 之后 ， 可以利 CountDownLatch 功 了

- CyclicBarrier (回) CyclicBarrier 作 会 后 会 下 动

  CyclicBarrier初 化 个 ， 后 了CyclicBarrier.await() 入 到了 个 ， 入 唤

CyclicBarrier初 可 个Runnable 参 , Runnable任务在CyclicBarrier 到后, 其 唤 前

- Semaphore (信号) Semaphore synchronized 加 , 作 制 发 (允 义 同) , 单 synchronized 关 不了